НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Програмна реалізація таймграфера із автоматичним розпізнаванням несправностей годинникових механізмів. Підсистема обробки сигналів механізмів»

Виконав (-ла):
студент (-ка) IV курсу, групи ІС-91
Косюк Михайло Михайлович _____
Керівник:
Доцент кафедри ІСТ, к.т.н
Шимкович Володимир Миколайович _____
Рецензент:
проф. каф. ІПІ., д.т.н., проф.
Стеценко Інна Вячеславівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Косюку Михайлу Михайловичу

1. Тема проєкту «Програмна реалізація таймграфера із автоматичним розпізнаванням несправностей годинникових механізмів. Підсистема обробки сигналів механізмів», керівник проєкту Шимкович Володимир Миколайович, к.т.н, доцент кафедри ІСТ, затверджені наказом по університету від «31» травня 2023 р. № 2101-с.

2. Термін подання студентом проєкту: «12» червня 2023 р.

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки:

1. Загальні положення: опис предметного середовища, огляд наявних аналогів та постановка задачі.
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис бази даних.
3. Математичне забезпечення: фізична модель, змістовна постановка задачі, математична постановка задачі, обґрунтування методу розв’язання та опис методу розв’язання.

4. Програмне забезпечення: Засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення та інфраструктура програмного забезпечення.
5. Технологічний розділ: керівництво користувача та тестування програмного продукту.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

1. Структурна схема варіантів використання
2. Структурна схема потоку даних
3. Структурна схема інфраструктури
4. Структурна схема класів програмного забезпечення
5. Структурна схема архітектурних компонентів
6. Структурна схема послідовностей

6. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	01.04.2023	
2.	Аналіз предметної області	21.04.2023	
3.	Формулювання вимог до системи	28.04.2023	
4.	Проектування архітектури системи	05.05.2023	
5.	Розробка інформаційного забезпечення	07.05.2023	
6.	Постановка математичної задачі	09.05.2023	
7.	Розробка програмного забезпечення	16.05.2023	
8.	Виконання графічних матеріалів	21.05.2023	
9.	Офромлення пояснювальної записки	25.05.2023	
10.	Подання ДП на попередній захист	29.05.2023	
11.	Подання ДП на основний захист	19.06.2023	

Студент Михайло КОСЮК

Керівник Володимир ШИМКОВИЧ

АНОТАЦІЯ

Косюк М.М. Програмна реалізація таймграфера із автоматичним розпізнаванням несправностей годинникових механізмів. Підсистема обробки сигналів механізмів. КПІ ім. Ігоря Сікорського, Київ, 2023.

Проект містить 90 с. тексту, 53 рисунків, 15 таблиць, посилання на 34 літературні джерела, додатки та 6 конструкторських документів.

Ключові слова: годинниковий механізм, мобільний пристрій, наручний годинник, таймграфер, точність годинника.

Об’єктом розробки є програмна реалізація інструментарію таймграфера.

Метою створення системи є надання функціоналу таймграфера на мобільному пристрої, при чому система має бути крос-платформеною, та створення цифрового формату даних вихідного результату таймграфера.

У дипломному проекті розроблено інформаційну систему інструментарію таймграфера у формі веб-додатку. Особливу увагу приділено дослідженню фізичної моделі механізму годинника та дослідженню звукового сигналу роботи механізму швейцарського спуску. Також розробка концентрується на використанні стандартних браузерних API Javascript для роботи зі звуком.

Розроблена система може бути корисною для швидкої поверхневої оцінки стану годинникових механізмів, а дослідження можуть бути взяті за основу для подальшого шлибшого вивчення звукового сигналу годинника.

ABSTRACT

M.M. Kosiuk Program implementation of timegrapher with automatic watch mechanism malfunction recognition. Subsystem of mechanism signals processing. Igor Sikorsky KPI, Kyiv, 2023.

The project contains 90 pages of text, 53 figures, 15 tables, 34 links to literature sources, annexes and 6 design documents.

Keywords: mobile device, timegrapher, watch accuracy, watch mechanism, wristwatch.

The object of development is the programmatic implementation of timegrapher functionality.

The purpose of system development is to provide functionality of a timgrapher on mobile devices while the system should be cross-platform, and to create a standard digital data format of timegrapher output.

The information system of timgrapher tooling was developed in course of the project. Special attention was paid to the research of watch mechanism's physical model and to the research of swiss lever escapement sound signals. As well, the development is focused on the usage of standard browser Javascript APIs for sound processing.

The developed system may be useful for fast initial assessement of watch mechanism status and the research results can be used as a basis fo further and deeper analysis of the watch sound signals.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІС91.010БАК.004 ПЗ	Пояснювальна записка	88		
6	А3	ІС91.010БАК.004 Д1	Структурна схема	1		
7			варіантів використання			
8	А3	ІС91.010БАК.004 Д2	Структурна схема	1		
9			Потоку даних			
10	А3	ІС91.010БАК.004 Д3	Структурна схема	1		
11			інфраструктури			
12	А3	ІС91.010БАК.004 Д4	Структурна схема класів	1		
13			програмного забезпечення			
14	А3	ІС91.010БАК.004 Д5	Структурна схема	1		
15			Архітектурних компонентів			
16	А3	ІС91.010БАК.004 Д6	Структурна схема	2		
17			послідовностей			
18						
19						
20						
21						
22						
23						
24						
25						
27						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІС91.010БАК.004 ТП		
Розроб.		Косюк М.М.			Програмна реалізація таймграфера із автоматичним розпізнаванням несправностей годинникових механізмів. Підсистема обробки сигналів механізмів. Відомість проєкту	Літ.	Аркуш	Аркушів
Керівн.		Шимкович В.М.				т	1	1
						КПІ ім. ІгоряСікорського Група ІС-91		
Затв.								

Пояснювальна записка
до дипломного проєкту
на тему: «Програмна реалізація таймграфера із автоматичним розпізнаванням несправностей годинникових механізмів. Підсистема обробки сигналів механізмів»

Київ – 2023

ЗМІСТ

ВСТУП

За декілька останніх років ринок наручних годинників переживає дуже стрімкий зріст, який сягає 6% середньорічного темпу приросту вже зараз[2]. Капітал індустрії складав 62 мільярди доларів США в 2020 році і має прогнози досягти 100 мільярдів до 2025 року[2]. При цьому найбільш стрімкий зріст спостерігається у сегменті механічних годинників, що спричинено в основному перенасиченням ринку персональної електроніки та правильними стратегіями компаній, що виробляють механічні годинники, на сьогоднішній день їх об'єми вироблення вимірюються мільйонами екземплярів щорічно[1]. Механічні годинники також мають велику вагу у ролі предмету статусу. На відміну від кварцевих або електронних годинників, механічні мають достатньо простий принцип роботи, а найголовніше зазвичай є в декілька разів дорожчими. У зв'язку з цим обслуговування та полагодження механічних годинників завжди мало місце, а сьогодні потреба у цьому зростає разом із ринком. При цьому для навіть найпростіших перевірок механічних механізмів необхідні спеціальні достатньо дорогі прилади, такі як таймграфер.

Об'єктом даної роботи є таймграфер – основний інструмент, яким користуються годинникові майстри для полагодження механізмів. Через традиційнсть та консервативність галузі механічних годинників та стрімкий зріст їх кількості, кількість майстрів не встигає зростати достатніми темпами, тому має місце факт самостійного огляду, а в деяких випадках і полагодження механізмів власниками. В останній час також спостерігається зріст сегменту уживаних годинників, де таймграфер відіграє ключову роль у перевірці механізму покупцем[1]. У зв'язку з цим розробка доступної альтернативи функціонала таймграфера набуває актуальності, що і є предметом роботи. За декілька останніх років були створені мобільні та десктопні додатки для покриття цієї потреби, але їх кількість досі вимірюється одиницями та найчастіше вони вимагають оплати від користувачів, що зменшує їх значимість.

З цієї причини метою даної роботи є надання функціоналу таймграфера на мобільному пристрої у вигляді крос-платформеної системи із веб-інтерфейсом, а

також створення стандартного цифрового формату даних вихідного результату таймграфера для простої інтеграції із іншими системами.

У результаті виконання роботи було розроблено систему, яка дозволяє користуватись основним функціоналом таймграфера, використовуючи для цього наявне апаратне забезпечення мобільного пристрою користувача. Клієнтська сторона система виконана у формі веб-додатку, що дозволяє користувачу не встановлювати ніякого програмного забезпечення на пристрій, а просто отримати доступ до додатку відкривши його у браузері. Система візуалізує та зберігає результати роботи. При цьому точність вимірювань хоч і не є ідеальною, але є достатньою для покриття основних задач. Враховуючи описане вище, можна сказати, що розроблений інструмент на практиці підійде для початкового швидкого огляду стану годинникового механізму, який може забезпечити як необхідні дані для проведення певних полагоджень власником, так і для прийнятті рішень у купівлі та продажі.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища

Предметним середовищем даної роботи є індустрії наручних годинників. На сьогоднішній день дослідження ринку показують стрімке зростання популярності механічних годинників. Це відображається як у кількості вироблених екземплярів так і в розмірі долі ринку[1]. Водночас механічні годинники, як і інші суто механічні прилади, добре піддаються ремонту та полагодженню, також як і всі механічні прилади, годинники потребують обслуговування, в ідеалі кожні 4-5 років[3]. При цьому для проведення навіть найбільш основних перевірок та найпростіших полагоджень необхідні дорогі спеціалізовані інструменти, найголовнішим з яких є таймграфер. Таймграфер також є важливим інструментом при придбанні уживаних годинників, що є досить поширеним у сегменті механічних годинників. На жаль, сьогодні кількість сервісів, що мають такі інструменти не встигає розвиватись за обсягом ринку, а придбання власного інструменту для більшості власників механічних годинників може видатись занадто дорогим, беручи до уваги наскільки рідко він буде використовуватись.

1.1.1. Опис процесу діяльності

Процесом діяльності, що досліджується в цій роботі є робота таймграфера. Таймграфер – це прибор який дозволяє візуалізувати роботу годинникового механізма, на основі звукових сигналів, що створюються в результаті взаємодії частин годинникового механізму а саме колеса балансу та важеля анкерного спуску, а також обраховувати основні характеристики механізму, такі як:

- *відхилення ходу* (rate deviation), характеристика, що описує наскільки «спішить» або «запізнюється годинник», зазвичай відображається у секундах на день[4];

- *помилка такту* (beat error), характеристика, що відображає різницю між прокручуванням колеса балансу по годинниковій та проти годинникової стрілки, ці два рухи також називаються «тік» і «так» в годинниковій справі, різниця

вимірюється в мілісекундах[4], візуалізацію колеса балансу та помилки такту можна побачити на рисунку 1.1;

- *амплітуда* (amplitude), характеристика, що відображає амплітуду прокручування колеса балансу в градусах, амплітуда є індивідуальною характеристикою для кожного механізму, тому для її аналізу потрібно чітке розуміння про те який механізм перевіряється, амплітуда вимірюється у кутових градусах[4], візуалізацію колеса балансу та амплітуди можна побачити на рисунку 1.2.

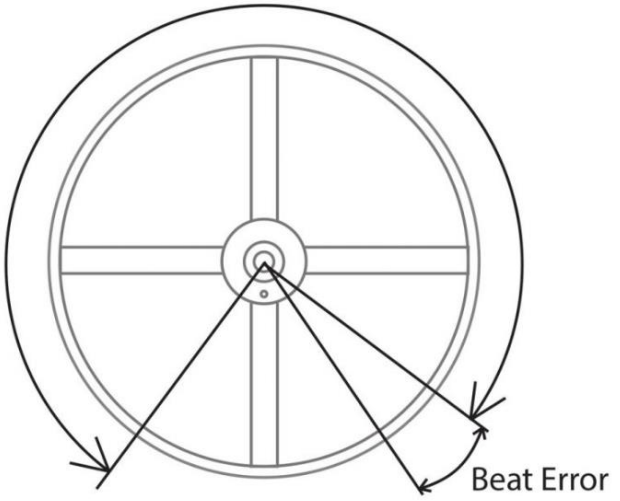

Рисунок 1.1 – Помилка такту

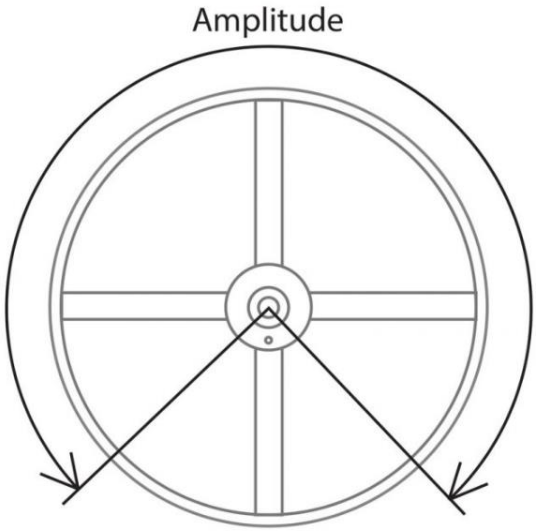

Рисунок 1.2 – Амплітуда

Реалізація таймграфера в даній роботі буде концентруватись тільки на обрахуванні відхилення ходу та помилки ходу. По причині вимоги дуже великої точності звукового сигналу для обчислення амплітуди та її неоднозначність, ця характеристика виключена із роботи.

Графік таймграфера на пристроях виглядає як лінії із набору точок, які можуть здатись сильно переривистими, але це зумовлено тільки невеликою роздільною здатністю екранів, що використовуються у таймграферах. Приклад графіка можна побачити на рисунку 1.3.

Рисунок 1.3 – Приклад графіку таймграфера

1.1.2. Опис функціональної моделі

Основною дійовою особою системи буде користувач. Додатково система розрахована на інтеграцію з іншими інформаційними системами у ролі джерела даних вимірювань. Функціонал системи матиме прямолінійну та майже повністю послідовну модель. Тобто система буде взаємодіяти із користувачем за допомогою певного лінійного алгоритму, проводячи користувача через набір кроків.

Варіанти використання системи перераховані у таблиці 1.1.

Таблиця 1.1 – Варіанти використання системи

Актор	*Варіант використання*	*Опис дії варіанта використання*
користувач	Аутентифікація	Аутентифікація користувача через стороннього провайдера
	Ручне введення частоти ходу механізму	Введення параметру частоти ходу механізму через графічний інтерфейс користувача
	Автоматичне визначення частоти ходу механізму	Визначення параметру частоти ходу механізму системою
	Запис звукових сигналів роботи механізму	Запис звукового сигналу із клієнтської сторони, використовуючи стандартний мікрофон мобільного пристрою
	Експорт запису у стандартному форматі	Експорт запису у стандартному форматі за допомогою завантаження файлу, або у формі URL

	Підрахунок характеристик механізму	Підрахунок системою характеристик помилки такту та відхилення ходу системою й виведення користувачу
	Виведення графіків роботи механізму	Виведення графіків хвильової форми та візуалізації роботи механізму, маніпулювання графіками
	Прослуховування запису роботи механізму	Прослуховування звукової доріжки запису роботи механізму

Кінець таблиці 1.1

Актор	*Варіант використання*	*Опис дії варіанта використання*
користувач	Введення метаданих	Введення набору додаткових метаданих до виміру через графічний інтерфейс користувача
	Перегляд записів	Перегляд списку записів
	Пошук записів	Пошук записів за тегами
стороння система	Отримання запису роботи механізму	Отримання запису у стандартному форматі через веб-інтерфейс

Діаграму варіантів використання системи подано у графічному матеріалі.

Функціональні вимоги створені на основі описаних вище варіантів використання системи наведені у таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги

Варіант використання	*Функціональна вимога*	*Пріоритет*
Аутентифікація	1. Система дає можливість пройти аутентифікацію за допомогою акаунта Google.	Високий
	1.1. Система розпізнає аутентифікованих користувачів	
	1.2. Система верифікує аутентифікацію користувача	

Ручне введення частоти ходу механізму	2. Система надає можливість ввести частоту ходу механізму.	Низький
	2.1. Система відображає поле для введення цілого числа	Низький
	2.2. Система перевіряє величину введеного значення за набором стандартних значень частоти.	Низький

Продовження таблиці 1.2

Варіант використання	*Функціональна вимога*	*Пріоритет*
Автоматичне визначення частоти ходу механізму	3. Система самостійно визначає значення частоти механізму.	Середній
	3.1. Система відображає поле із підрахованим значенням	Середній
	3.2. Система вибирає найближче значення із набору стандартних значень частоти	Середній
Запис звукових сигналів роботи механізму	4. Система проводить запис звукового сигналу	Високий
	4.1. Система відображає кнопку для початку запису	Високий
	4.3. Система конвертує звуковий запис у стандартний формат	Високий
	4.4. Система зберігає форматований запис	Високий
	4.5. Система відображає текстові поля для додавання метаданих до запису	Низький
Експорт запису у стандартному форматі	5. Система дозволяє експортувати збережений форматований запис	Середній
	5.1. Система відображає кнопку для завантаження файлу із записом	Середній

	5.2. Система надає URL для отримання форматованих даних	Високий
Підрахунок характеристик механізму	6. Система підраховує характеристики	Середній
	6.1. Система підраховує відхилення ходу та виводить користувачу.	Середній
	6.2. Система підраховує помилку такту та виводить користувачу.	Середній

Продовження таблиці 1.2

Варіант використання	*Функціональна вимога*	*Пріоритет*
Виведення графіків роботи механізму	7. Система виводить зображення графіків роботи механізму.	Високий
	7.1. Система виводить графік таймграфера	Високий
	7.2. Система виводить хвильову форму звукового запису	Середній
	7.2.1. Система відображає кнопки для переміщення по хвильовій формі	Низький
Прослуховування запису роботи механізму	8. Система надає можливість прослухати звуковий запис роботи механізму	Середній
	8.1. Система відображає набір кнопок для відтворення та перемотки запису	Середній
Введення метаданих	9. Система надає можливість введення метаданих у вигляді тегів	Низький
	9.1. Система відображає набір полів для введення тегів	
	9.2. Система пропонує існуючі теги	
	9.3. Система обмежує максимальну кількість тегів до 5	

	9.4. Система зберігає теги до бази даних разом із записом, а також нові теги у разі їх наявності.	
Перегляд записів	10. Система надає можливість переглядати всі існуючі записи	Середній
	10.1. Система відображає список записів із тегами та датою	
	10.2. Система відображає ряд кнопок для переходу між сторінками списку	

Кінець таблиці 1.2

Варіант використання	*Функціональна вимога*	*Пріоритет*
Пошук записів	11. Система надає можливість шукати записи за тегами.	Низький
	11.1. Система відображає поле для множинного введення шуканих тегів	
	11.2. Система відображає тільки записи, що мають шукані теги	
Отримання запису роботи механізму	12. Система надає ендпойнт веб-інтерфейсу для отримання форматованого запису	Високий
	12.1. Система надає можливість додавання ідентифікатору запису до ендпойнту для отримання конкретного запису	Високий

1.2. Огляд наявних аналогів

Для пошуку аналогів програмного забезпечення до системи, що розробляється в цій роботі було використано такі джерела як:

- маркетплейс додатків для ПК Microsoft Store;
- маркетплейси мобільних додатків Google Play Market та App Store;

- мережа інтернету як джерело потенційних веб-додатків.

В результаті проведеного дослідження в цілому можна сказати, що на сьогоднішній день розробка програмних інструментів для використання у предметній області механічних годинників досі знаходиться на ранньому етапі, скоріше за все це пов’язано із різким зростом популярності механічних годинників за останні 2-3 роки[1], за яким ще не встигли розвинутись як програмне забезпечення, так і інформаційне забезпечення. Це також може бути пояснюється відносною ізольованістю галузі та невеликого відсотка молодих людей у ній, на підтвердження цьому факт того, що певні знайдені аналоги були розроблені більше 10 років тому, але так і не отримали широкого використання.

Всього було знайдено близько 15 потенційних аналогів, що є дуже малою кількістю, при чому тільки 4 із них заслуговують окремої уваги, оскільки всі інші досі не є повністю функціонуючими інструментами за різними причинами.

Timegrapher X

Ймовірно найкращим існуючим аналогом є мобільний додаток для IOS Tmegrapher X. Додаток є найпопулярнішим серед годинникових майстрів, які мають можливість порівняти його роботу із реальним приладом.

Переваги:

- додаток має ймовірно найкращу точність вимірювань серед аналогів;
- додаток вимірює усі основні характеристики механізму, включаючи амплітуду;
- додаток відображає усі необхідні графіки, такі як хвильова форма звукового сигналу, візуалізацію таймграфера та графік довоготривалого аналізу механізму.

Недоліки:

- додаток підтримується тільки на платформі IOS;
- додаток має дуже високу ціну у розмірі 36 USD;
- точність додатка зумовлена апаратною оптимізацією, додаток оптимізовано до роботи із оригінальною мобільною гарнітурою Apple, тобто робота заточена під конкретний мікрофон.

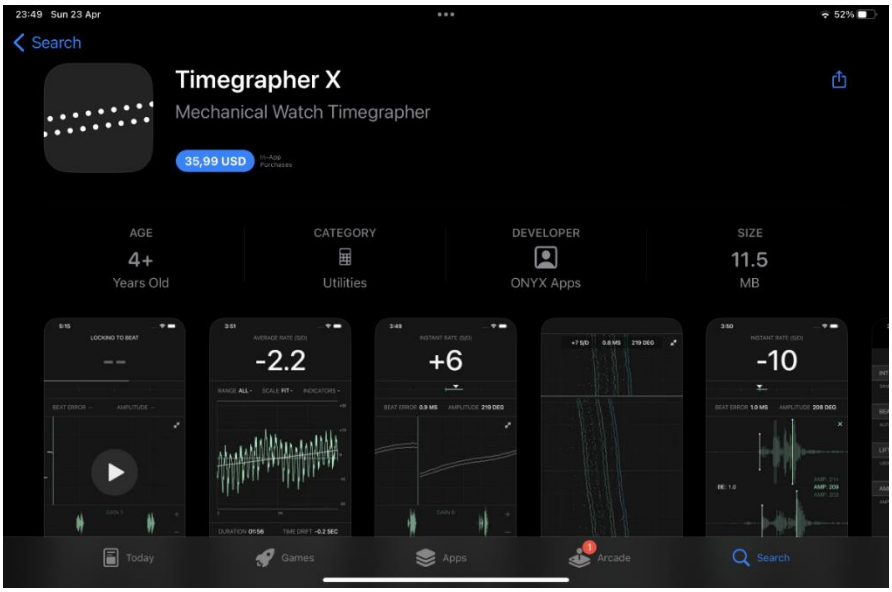

Рисунок 1.4 – Огляд Timegrapher X в App Store

Watch Tuner Timegrapher

Ще один мобільний додаток для IOS, має посередні можливості.

Переваги:

- додаток має достатню точність для його практичного застосування;
- додаток вимірює усі основні характеристики механізму, включаючи амплітуду;
- додаток має широкі можливості по налаштуванню, зокрема введення характеристики куту нахилу механізму при вимірюванні.

Недоліки:

- додаток підтримується тільки на платформі IOS;
- додаток є платним і має ціну 6 USD;
- відгуки про точність додатку дуже неоднорідні.

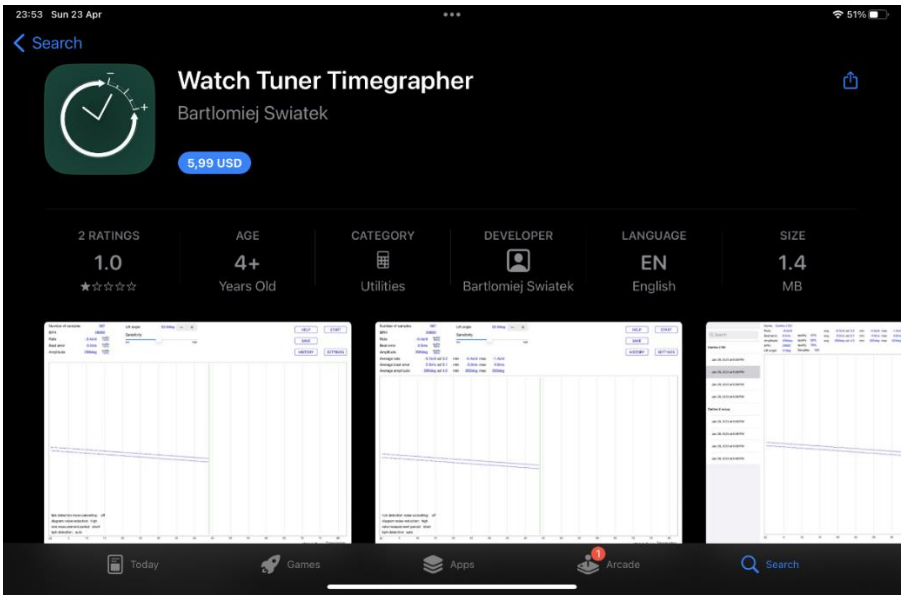

Рисунок 1.5 – Огляд Watch Tuner Timegrapher в App Store

Watch Accuracy Meter

Найпопулярніший мобільний додаток для платформи Android, має посередні можливості, але є повністю безкоштовним.

Переваги:

- додаток має достатню точність для його практичного застосування;
- додаток має необхідний набір налаштувань;
- додаток вимірює основні характеристики механізму;
- додаток має унікальну можливість порівняльного графіку вимірювань одного механізму в різних позиціях (нормальна, перевернута і т.д.), що є просунутим методом огляду стану механізму;
- додаток є повністю безкоштовним.

Недоліки:

- додаток підтримується тільки на платформі Android;
- додаток не вимірює амплітуду механізму.

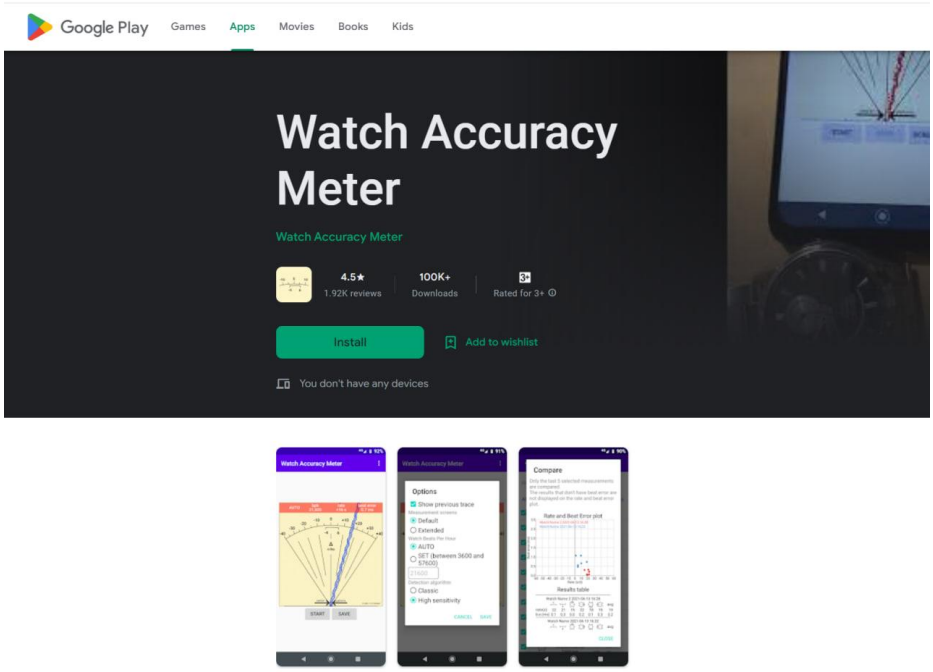

Рисунок 1.6 – Огляд Watch Accuracy Meter в Google Play Market

Timegrapher

Другий гідний мобільний додаток для платформи Android, має схожі характеристики до попереднього аналога, але є частково платним.

Переваги:

- додаток має достатню точність для його практичного застосування;
- додаток має необхідний набір налаштувань;
- додаток вимірює усі основні характеристики механізму, включаючи амплітуду;
- додаток має можливість додавання позиції механізму при збереженні результату вимірювань.

Недоліки:

- додаток підтримується тільки на платформі Android;
- безкоштовна версія додатка має дуже обмежений функціонал, а весь додатковий функціонал купується окремо.

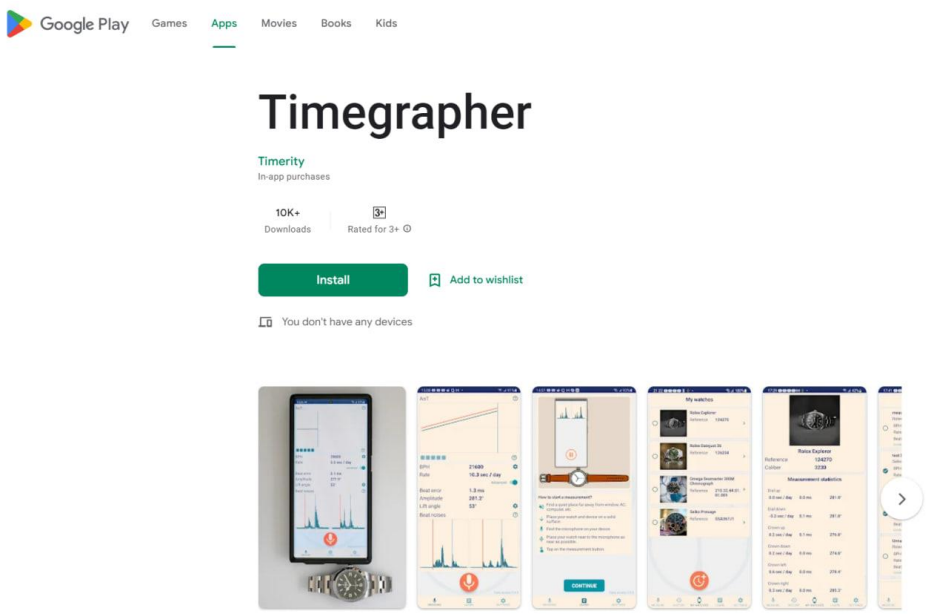

Рисунок 1.7 – Огляд Timegrapher в Google Play Market

На основі даних про найкращі аналоги проведемо їх порівняльний аналіз за наступними параметрами:

- Т - Точність (по шкалі від 1 до 5);
- ПФТ - Повнота функціоналу таймграфера (по шкалі від 1 до 5);
- ДМ - Додаткові можливості (по шкалі від 1 до 5);
- Д - Доступність (матеріальна доступність по шкалі від 1 до 5);
- КП - Крос-платформеність (0 – відсутня, 2 - присутня).

Результати наведені у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика аналогів системи

Параметр / Аналог	Т	ПФТ	ДМ	Д	КП	Σ
Timegrapher X	5+	5	2	1	0	13
Watch Tuner Timegrapher	3	5	2	2	0	12
Watch Accuracy Meter	3	4	3	5	0	15
Timegrapher	4	5	2	3	0	14

За таблицею видно, що всі додатки достатньо близькі в цілому, але сильно відрізняються в основному в доступності та точності. Слід відмітити, що за результатами аналізу не існує ні одного крос-платформеного додатку.

Найкращим аналогом загалом є Watch Accuracy Meter. Таким чином, можливістю для покращення є створення безкоштовного крос-платформеного додатку, який за точністю повинен не сильно відрізнятись від Watch Accuracy Meter.

1.3. Постановка задачі

Враховуючи проведений аналіз наявних аналогів, основною задачею роботи є створення таймграфера, який може оперувати на рівні не гіршому за конкурентів у формі веб-додатку під короткою назвою «OpenTimegrapher».

1.3.1. Призначення розробки

Система реалізує таймграфер, що є спеціалізованим інструментом для годинникових майстрів та в класичному вигляді поєднує апаратне забезпечення та інтегроване програмне забезпечення в одній системі. «OpenTimegrapher» надає функціонал таймграфера (як зчитування звукових сигналів годинникового механізму, візуалізація роботи механізму та визначення його певних характеристик), поєднуючи апаратне забезпечення мобільних пристроїв та програмне забезпечення створене на основі крос-платформених технологій, що дає можливість використання таймграфера широкому колу користувачів у формі веб-додатку та зменшує потребу в спеціальному апаратному обладнанні.

Система також реалізує стандартний формат даних для представлення результату роботи таймграфера, що дозволяє візуалізувати та оброблювати ці дані в інших системах, наприклад використання цих даних в якості датасету для навчання нейронних мереж.

1.3.2. Визначення цілей та задач розробки

Головною ціллю створення системи є надання функціоналу таймграфера на мобільному пристрої, при чому система має бути крос-платформеною у формі веб-додатку, та створення цифрового формату даних вихідного результату

таймграфера. Надалі називатимемо цей формат даних, що описують роботу годинникового механізму сигнатурою механізму.

Для досягнення цілі необхідно вирішити наступні задачі:

- створення вебзастосунку клієнтської сторони для проведення вимірювань;
- використання стандартного користувацького мікрофону для точних вимірювань, перетворення та покращення сигналу;
- створення простого стандартного дискретного формату даних результату вимірювань – сигнатури механізмів;
- конвертація звукового сигналу у стандартний формат даних та його зберігання;
- визначення точності ходу механізму за допомогою результуючих даних;
- створення веб-інтерфейсу для представлення функціональності системи клієнтському веб-додатку;
- створення мульти-контейнерного образу усіх компонентів системи для простого запуску у різних операційних системах.

Висновок до розділу

В цьому розділі було повністю описане предметне середовище роботи, описано функціональну модель та процес її діяльності. Було сформовано сценарії використання системи та функціональні вимоги до них. Також було розглянуто існуючі аналоги системи для різних платформ та проведено їх порівняльний аналіз. За результатами аналізу було чітко визначено можливості покращення системи в порівнянні із аналогами та поставлено цілі та задачі розробки. Додатково було описано призначення розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Для спрощення опису потоку інформації у системі відобразимо основні вузли та дані за допомогою діаграми Data-flow за нотацією Йордона-Де-Марко[5], яку можна побачити у графічному матеріалі та на рисунку 2.1.

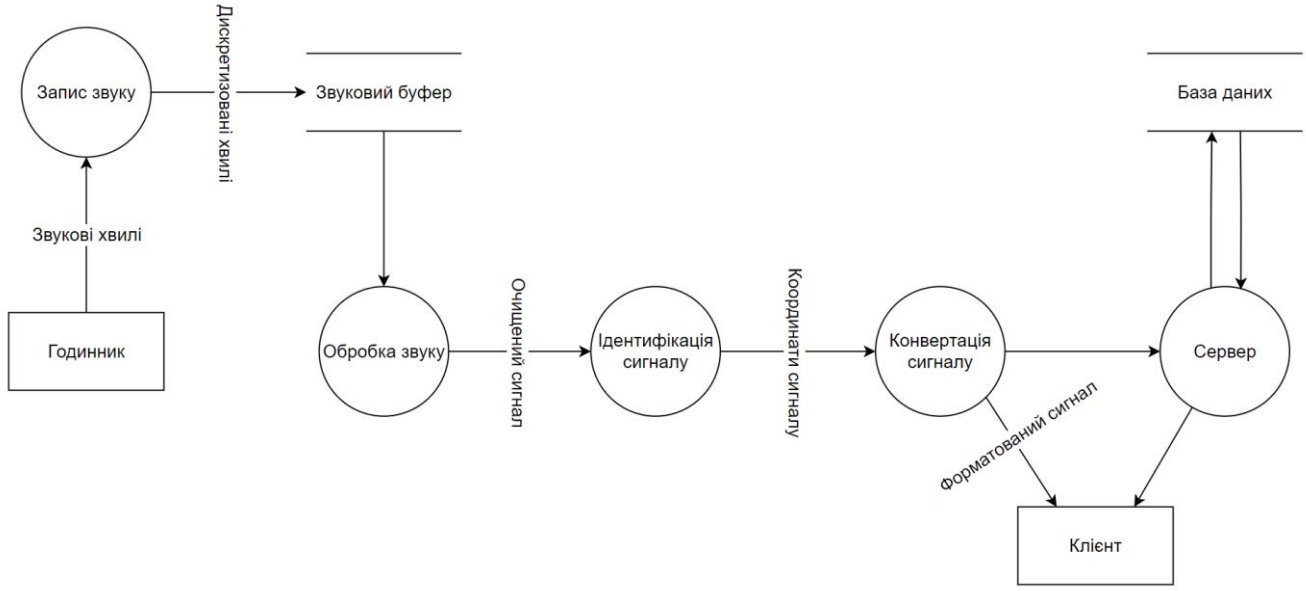

Рисунок 2.1 – Діаграма Data-flow

Діаграма призначена для відображення потоку даних у системі, вона не бере до уваги архітектурний тип компонентів або інфраструктуру системи, натомість повністю зосереджується на даних, їх передачі та перетворенні. Далі описано вхідні та вихідні дані для кожного вузла діаграми. Розбиття даних на вхідні та вихідні є умовним у контексті системи, по діаграмі видно, що одні й ті самі дані можуть бути вихідними для одного вузла та вхідними для іншого. Тому опишемо усі дані між вузлами як вхідні, а як вихідні дані опишемо тільки ті, які представляють кінцевий результат і не обробляються в подальших процесах. Це допоможе не дублювати дані у вхідних та вихідних.

2.1 Вхідні дані

Звукові хвилі

Звукові хвилі є першою та безпосередньою інформацією яку отримує на вхід система. Як відомо звукові хвилі є видом механічних хвиль, які можуть поширюватись у достатньо щільних середовищах, в даному випадку - у повітрі.

Звукові хвилі є аналоговим (безперервним) сигналом, який не можна напряму представити у вигляді певних дискретних даних, вони можуть походити від будь-яких фізичних об’єктів з навколишнього середовища, у випадку даної системи цільовим фізичним об’єктом є частина механізму годинника, яка створює характерний звук «тікання».

У зв’язку із реальною фізичною природою сигналу, абсолютним чином ізолювати корисні звукові хвилі від будь-яких навколишніх шумів або звуків – просто неможливо, що є основною складністю в роботі зі звуковими сигналами та з усіма реальними сигналами в цілому. Також як відомо хвилі мають властивість інтерференції, що означає можливість накладання хвиль різних частот одна на одну. Беручи до уваги усе вище сказане вхідний сигнал можна приблизно представити на рисунку 2.2. у вигляді неперервної хвилі, яка є сумою хвиль різних частот.

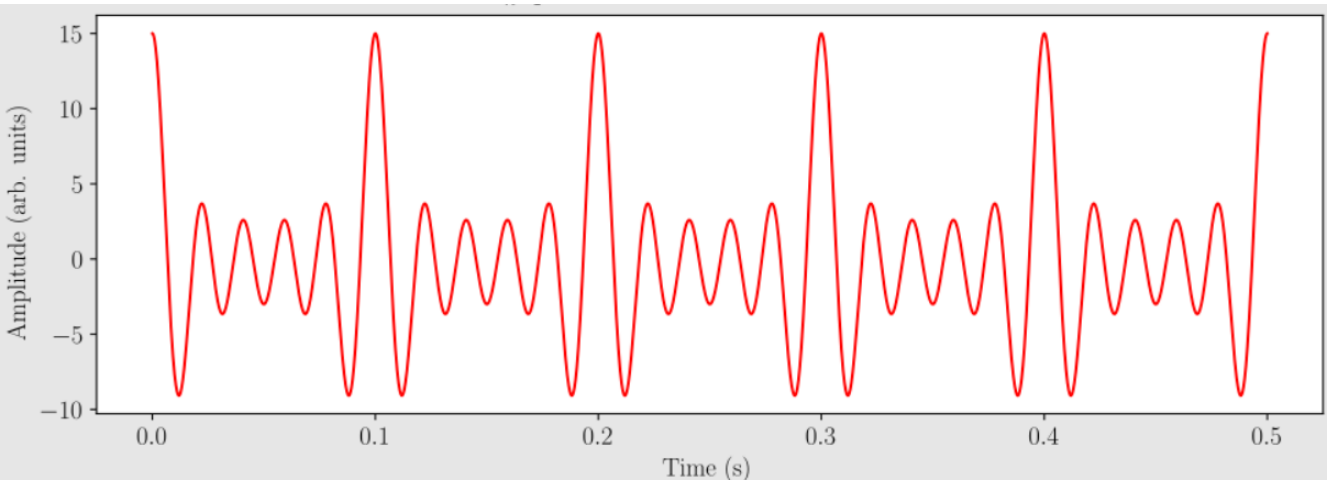

Рисунок 2.2 – Схематичне представлення реальної звукової хвилі

Двома найважливішими характеристиками звукових хвиль є амплітуда, що результує у гучність звуку та частота, що результує у тембр звуку.

Вид даних: аналоговий сигнал.

Дискретизовані хвилі

Очевидно, що обчислювальні машини, які мають цифрову природу не можуть напряму взаємодіяти із аналоговим сигналом, для виправлення цього використовується цифровий мікрофон та процес дискретизації хвиль. Мікрофон записує записує звукову хвилю з певною достатньо високою частотою, в

середньому сучасні мікрофони записують звук із частотою 48 кГц, тобто кожну секунду мікрофон створює та записує 48000 значень амплітуди (гучності) звуку. Ці значення також називаються семплами (англ. sample), а характеристика частоти запису – семпл-рейтом (англ. sample rate), або частотою дискретизації. Таким чином отримується інформація яку можна обробляти за допомогою дискретних методів, представлення дискретизованої хвилі побачити на рисунку 3.3.

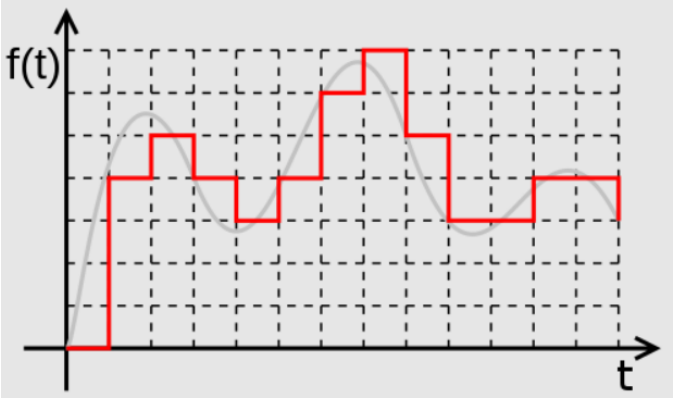

Рисунок 2.3 – Схематичне представлення звукової хвилі

Далі ці дані зберігаються в умовний буфер (файл).

Вид даних: масив числових значень амплітуди a_n – [a_1, a_2, a_3 … a_n].

Очищений сигнал

Наступним видом даних є очищений сигнал, він не відрізняється за своєю структурою від дискретизованих хвиль, але має більш значний корисний сигнал за допомогою пропускання первинного сигналу через ряд частотних фільтрів.

Вид даних: масив числових значень амплітуди a_n – [a_1, a_2, a_3 … a_n].

Координати сигналу

Після отримання даних за якими достатньо точно можна виділити корисний сигнал алгоритмічним чином, на стадії ідентифікації сигналу визначаються часові координати початку та кінця корисного сигналу, тобто «тіків» для даної задачі.

Для задоволення потреб системи достатньо визначення двох точок: початок «тіка», та його кінець.

Вид даних: масив пар координат значень s_m та e_m, що відповідають початку та кінцю «тіка» – [[s_1, e_1], [s_2, e_2], [s_3, e_3] … [s_m, e_m]].

Додаткові вхідні параметри

Окрім основних даних система також може приймати на вхід певні опціональні додаткові параметри:

- Частота ходу механізму (beat rate) числове значення із фіксованого набору чисел beatRate є {18000, 19800, 21600, 28800, 36000}.

- Метадані у вигляді множини тегів, що можуть мати як числові так і символьні значення. Прикладом тегу може бути бренд годинника.

2.2 Вихідні дані

Форматований сигнал

Останньою та вихідною формою даних у системі є форматований сигнал. Вихідний формат даних містить:

- Координати точок графіку таймграфера, що був представлений в розділі 1. Точки таймграфера несуть значення відхилення інтервалу між двома тіками у порівнянні з номінальним інтервалом у більшу чи меншу сторону;

- дві інтегральних характеристиками цього графіка rate error та beat error, які також були представлені у розділі 1;

- значення частоти ходу механізму введене вручну або визначене автоматично;

- додатково формат містить введені метадані

Цільовий формат може напряму відображатись на пристрої клієнта та зберігатися у базу даних.

Вид даних: приклад формату вихідних даних можна побачити на рисунку 2.4.

```
{
  "plot": [0.1, -0.02, 0.08, 0.01, 0.13, -0.01],
  "beatRate": 21600,
  "rateError": -10,
  "beatError": 0.6,
  "metaTags": ["orient", "automatic", "f6922"]
}
```

Рисунок 2.4 – Приклад вихідного формату

Детальні алгоритми процесів перетврення даних можна знайти у розділі 3.

2.3 Опис структури бази даних

Система не оперує окремими сутностями і не потребує зв’язків між ними, натомість пріоритетною задачею є швидкий запис та зчитування даних неоднорідного формату. Для таких вимог найкраще підходить документна NoSQL база даних. Конкретно для системи було обрано Firestore, що є нативною документною NoSQL базою даних Google Cloud Platform.

Основними структурними одиницями документної бази даних є колекція, документ та поле документа, будь-які реляційні зв'язки між документами не підримуються самою базою даних, хоча можуть бути створені, тому представлення ER діаграм є значно простішим ніж для реляційних баз даних.

База даних системи буде містити дві колекції, її структуру можна побачити на рисунку 2.5.

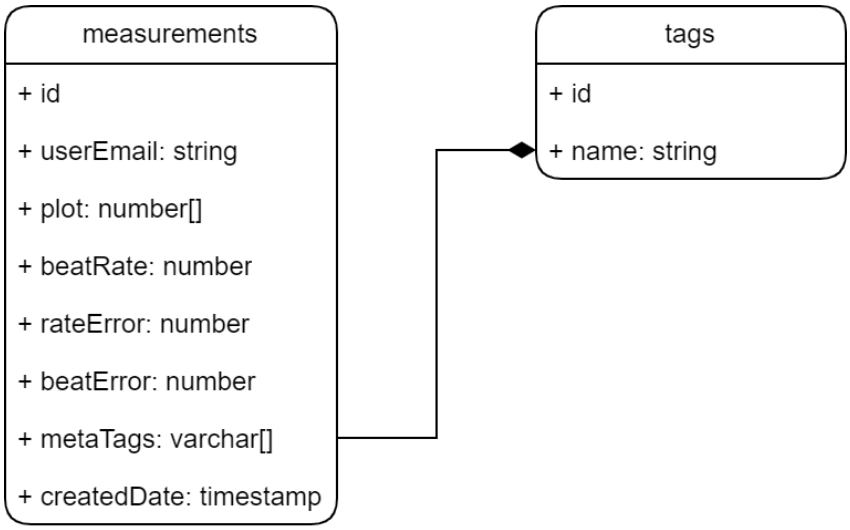

Рисунок 2.5 – Структура бази даних

База даних містить дві колекції:

- measurements – відповідає за зберігання записів основного вихідного формату системи із унікальним ідентифікатором та датою створення;
- tags – окрема колекція для зберігання унікальних тегів, основною задачею якої буде швидкий пошук тегів, поле *metaTags* колекції *measurments* може складатись тільки із тегів які існують в колекції *tags*.

Опис структури бази даних подано в таблиці 2.1.

Таблиця 2.1 – Опис структури бази даних

Колекція	*Поле*	*Тип даних*	*Пояснення*
measurments	id	рядок	Унікальний ідентифікатор
measurments	userEmail	рядок	Ідентифікатор користувача
measurments	plot	масив чисел	Список точок графіка
measurements	beatRate	число	Частота ходу механізму
measurements	rateError	число	Відхилення ходу механізму
measurements	beatError	число	Помилка такту механізму
measurements	metaTags	масив рядків	Список тегів запису
measurments	createdDate	timestamp	Момент створення запису у мілісекундах з початку епохи Unix.
tags	id	рядок	Унікальний ідентифікатор
tags	name	рядок	Текст тегу

Висновок до розділу

В цьому розділі було описано потік інформації у системі за допомогою діаграми Data flow, також було описано природу і вид даних перед кожним етапом їх обробки у вузлах системи. Було чітко визначено основний вихідний формат даних, а також описана структура бази даних із поясненням кожного елемента.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

В цьому розділі буде описано математичний апарат, використаний для вирішення основних задач системи. Перед тим як розглядати суто математичну сторону системи, має сенс розглянути фізичну модель, оскільки система напряму співпрацює із реальним фізичним об’єктом, не використовуючи ніяких абстрактних інтерфейсів для отримання вхідних даних.

3.1 Фізична модель

Цільовим фізичним об’єктом є механічний годинник, а точніше конкретна частина механізму годинника, яка створює характерний сигнал «тікання», що використовується системою.

Частина механізму, що створює сигнал називається спусковим механізмом, на сьогоднішній день існує до десятка основних видів годинникових спускових механізмів, які мають різну будову і відповідно – різний сигнал. Проте таймграфер створений для обробки сигналу одного конкретного виду спускового механізму, який називається Швейцарський анкерний спуск (англ. Swiss lever escapement)[6], що також є найрозповсюдженішим видом спускового механізму. Він складається із трьох основних частин, що можна побачити на рисунку 3.1:

- (1)колесо балансу (англ. Balance wheel) є безпосереднім осцилятором, тобто джерелом точних коливань;
- (2)анкерна виделка (англ. Fork of pallets) є важелем, що передає коливання колеса балансу до спуску та контролює спуск за допомогою (2.1)зубців-палет, та (2.2)списом на іншому кінці для приймання коливань;
- (3)спускове колесо (англ. Escape wheel) є храповим колесом, із зубцями які зчіплюються з палетами виделки для забезпечення дозованого вивільнення енергії.

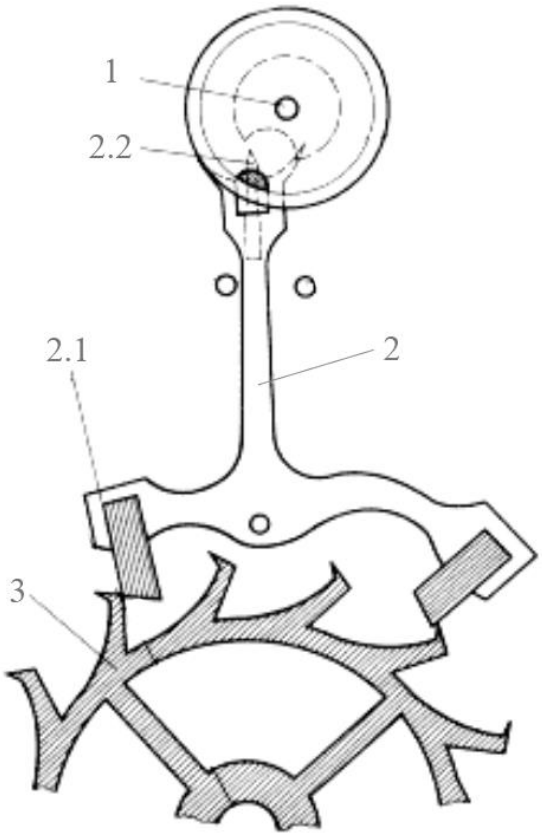

Рисунок 3.1 – Швейцарський анкерний спуск

Саме ці частини і створюють звук характерного «тікання» в процесі роботи механізму. Цей сигнал і є корисним сигналом таймграфера, зя яким він проводить вимірювання.

Природа цього сигналу є значно складнішою ніж здається на перший погляд. Один «тік», який може розрізнити слух людини насправді складається із трьох окремих звуків, що проходять за проміжок 15-30 мілісекунд і очевидно не розрізняються людиною, але їх розрізнення є невід'ємною частиною роботи таймграфера. На рисунку 3.2[7] представлено приклад хвильової форми звуку одного «тіка», де чітко видно що він складається із 3 звуків, кожен з яких напряму пов'язаний із роботою механізма.

Рисунок 3.2 – Приклад хвильової форми звуку «тіка»

Кожний із трьох звуків пов’язаний із певним зіткненням (тертям) між частинами спускового механізму. Перший звук спричинений ударом колеса балансу по спису анкерної виделки, як видно на рисунку 3.3[7]. Цей удар походить безпосередньо від джерела точних коливань, а отже є найточнішим за часом, тому саме цей звук використовується для основних обчислень в таймграфері. Також через відносно малий розмір частин механізму, які його спричиняють, цей звук найчастіше може бути найтихішим із трьох.

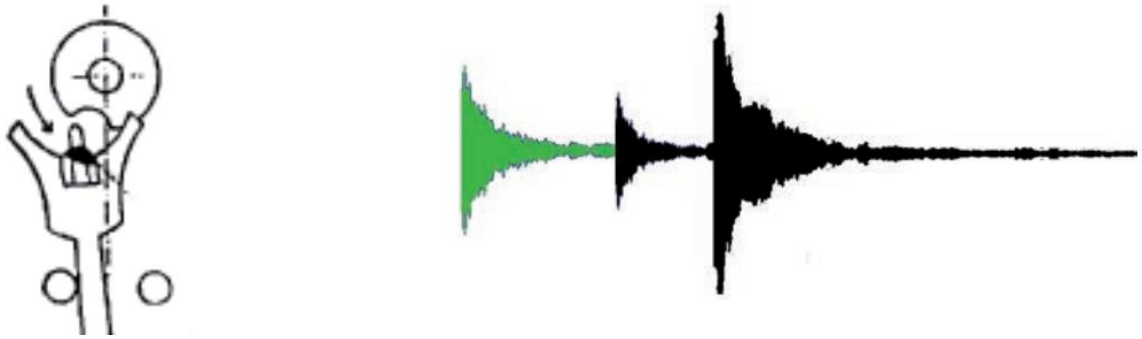

Рисунок 3.3 – Удар колеса балансу та списа анкерної виделки

Перший удар спричиняє рух важеля виделки вправо (відносно рисунка 3.1), що у свою чергу спричиняє вивільнення зубця спускового колеса від лівого (зачіпленого) палету виделки як видно на рисунку 3.4[7]. Цей звук не є ударом, а скоріше клацанням, що пов’язане із подовженим тертям елементів один об одного і різким зіскоком. Через свою природу цей звук є дуже неоднорідним і може відрізнятись навіть у двох почергових тіках, з цих причин він не використовується для вимірювань взагалі, але в силу своєї природи та розміру активних елементів, майже завжди є гучнішим за перший точний звук.

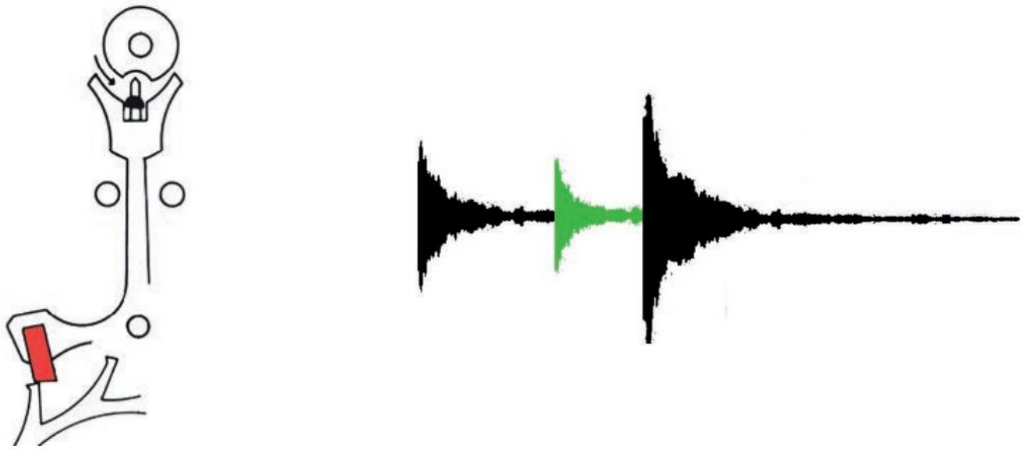

Рисунок 3.4 – Вивільнення зачіпленого палету від зубця спускового колеса

Вивільнення спускового колеса дозволяє йому продовжити рух у напрямку по годинниковій стрілці (відносно рисунку 3.1), в той же час правий вільний палет виделки опускається вниз, встаючи на рівень зубців спускового колеса, через дуже короткий час відбувається зіткнення зубця колеса та палету, палет ловить та фіксує колесо на льоту як видно на рисунку 3.5[7]. Завдяки тому, що зубець колеса отримує певне прискорення перед зіткненням із палетом, цей звук завжди є найгучнішим із трьох, тому саме за ним можна легко вирізнити сигнал із інших звуків.

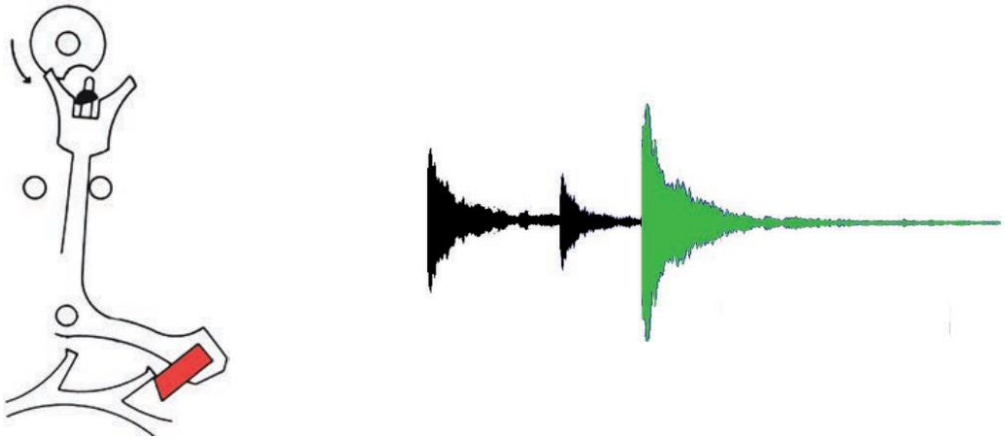

Рисунок 3.5 – Удар зубця спускового колеса та вільного палету виделки

Таким чином між двома координатами у сигналі, одна з яких використовується для вирізнення сигналу, а інша – для точного вимірювання, знаходиться неоднорідний за формою та гучністю звук, при цьому усі три звуки проходять у дуже короткий проміжок часу та є відносно дуже тихими.

Ще однією важливою характеристикою сигналу окрім його форми є його частотна характеристика, кожний тип природного звуку має різну частоту звукових хвиль. У випадку сигналу спускового механізму годинника частотна характеристика також є нетривіальною. По-перше матеріал активних елементів кожного із трьох звуків не є однаковим, зазвичай палети та спис анкерної виделки зроблені зі штучного рубіна, каменю, колесу балансу та спускове колеса зроблені з різних металевих сплавів. Отже матеріали першого звука відрізняються від матеріалів другого та третього звуків, а отже відрізняється і їх частотна характеристика, нехай навіть незначно.

Наступним фактором є ширина спектру звука, з цієї точки зору звуки діляться на гармонічні та шумоподібні. Енергія (амплітуда) гармонічних звуків зосереджена у відносно вузькому частотному спектрі, а отже виділяти такі звуки достатньо легко. З іншої сторони енергія шумоподібних звуків відносно рівномірно розбита по достатньо широкому спектру, або можна сказати – розбита на певну кількість гармонік. На прикладі людського голосу «А» є гармонічним звуком, а наприклад «Ц» є шумоподібним звуком[8]. Якщо симулювати звук тікання за допомогою звуків алфавіту, отримуємо щось наближене до «тссц», що очевидно є шумоподібним звуком, а отже розподілено по широкому частотному спектру, що ускладнює виділення сигналу.

У підсумку огляду фізичної моделі можна сказати, що задача ідентифікації сигналу для даного випадку є досить складною через саму природу сигналу, як частотну так і форму. Додаткової складності надає той факт, що сигнал походить не із готового обробленого запису, а із реального навколишнього середовища, яке гарантує наявність шумів.

3.2 Змістовна постановка задачі

Існує годинник із механічним ходом, механізм годинника має стандартний beat rate із відомого списку, тобто частоту коливань колеса балансу, яка може бути невідомою із початку.

Робота годинникового механізму створює періодичний звуковий сигнал «тікання», про який початково відома тільки його приблизна хвильова форма. Цей звуковий сигнал записується за допомогою мікрофону в умовах відносної, але не повної тиші.

За допомогою отриманого звукозапису необхідно побудувати графік таймграфера для даного механізму, який відображає відхилення кожного сигналу від еталонного значення, а також обчислити дві характеристики механізму – відхилення ходу (rate error) та помилку такту (beat error). При цьому побудований графік повинен мати не більше 1-2% пропущених точок, а в ідеалі пропущені сигнали мають бути відсутніми взагалі.

Для досягнення основної мети необхідно вирішити декілька важливих підзадач:

- підвищення рівня сигналу механізму у звукозаписі, що дозволить більш однозначно знаходити корисний сигнал;
- визначення частоти коливань механізму (beat rate);
- точна ідентифікація відрізків корисного сигналу у покращеному записі;
- точна ідентифікація місця початку сигналу.

3.3 Математична постановка задачі

Дано механічний годинник із невідомою частотою ходу (beat rate) $b \in B$, де B – множина стандартних частот ходу годинникових механізмів, що вимірюється у вібраціях за годину (VPH). $B = \{18000, 19800, 21600, 28800, 36000\}$. Також дано звуковий запис роботи механізму, про який відомий час запису t_g секунд та семпл рейт (sample rate) s_r семплів за секунду. Запис представлено у формі часової послідовності значень дискретизованого звукового сигналу r, де r_i – значення амплітуди семпла в момент часу t_i, i – порядковий номер елемента r. Значення будь-якого моменту часу t_i в мілісекундах можна розрахувати за формулою:

$$t_i = \frac{i}{s_r/1000} \tag{3.1}$$

Послідовність значень звукозапису можна представити як складну нелінійну функцію $f_r(i) = r_i$, задану табличним методом на відрізку $i \in [0 .. t_g \times s_r]$. Відомо, що механізм годинника створює періодичний звуковий сигнал через певний еталонний період T_b мілісекунд, який зумовлюється значенням b і може бути розрахований за формулою:

$$T_b = \frac{1000}{b/3600} \tag{3.2}$$

Значення T_b та будь-якого іншого реального часу, може бути переведене у відносні порядкові номери i використовуючи наступну формулу:

$$i_{Tb} = T_b \times s_r/1000 \quad (3.3)$$

Про сигнал також відома його форма, яка була описана у розділі 3.1 та представлена на рисунку 3.2. Мають місце наступні припущення про сигнал:

- максимальний проміжок часу усіх трьох звуків сигналу t_{sigmax} = 30 мс;
- максимальний час загасання третього звуку t_{amax} = 10 мс.

З точки зору функції f_r можна сказати, що існують невідомі відрізки $[i_j .. i_j + t_{sigmax}]$, де i_j – момент початку сигналу («тіка») з порядковим номером j, які розташовані через інтервал наближений до i_{Tb} та має місце наступне припущення:

- локальний максимальний екстремум $f_{r\ maxj}$ відповідає початку третього звуку сигналу. При чому таке припущення має місце тільки на зазначених відрізках, на інших відрізках припущення може бути як правдивим, так і хибним у випадку наявності значних шумів біля корисного сигналу.

Основною задачею є знаходження значень i_j, за їх допомогою потрібно знайти значення фактичних інтервалів між сигналами T_j, що можуть бути розраховані за формулою (використовуючи формулу 3.1):

$$T_j = \frac{(i_{j+1} - i_j)}{s_r/1000} \quad (3.4)$$

У підсумку необхідно створити послідовність $T_{\Delta j}$ відхилень періоду від еталонного, яка і є готовим графіком таймграфера, а також розрахувати дві інтегральні характеристики цієї послідовності:

- відхилення ходу (rate error) r_{err} за формулою:

$$r_{err} = (\sum_{j=0}^{n_j} T_{\Delta j})/n_j \quad (3.5)$$

- помилка такту (beat error) b_{err} за формулою:

$$b_{err} = (\sum_{j=1}^{n_j} T_{\Delta j} - T_{\Delta j-1})/n_j \quad (3.6)$$

3.4 Обґрунтування методу розв’язання

Як може бути видно, суто математична постановка задачі є досить складною та містить багато невідомих і відносно малу кількість припущень та даних, крім того основна досліджуваний сигнал є складною нелінійною функцією. Як відомо математичний аналіз таких функцій є дуже нетривіальною задачею і потребує глибоких та довгих досліджень функції, так само популярний метод апроксимації математичних функцій за допомогою нейронних мереж[9], на жаль, не є ефективним в даному випадку, оскільки згадана функція може дуже сильно відрізнятись для різних годинникових механізмів, можна навіть сказати, що кожен механізм буде мати свою власну унікальну функцію, тобто сигнатуру.

Враховуючи вищезгадане, розробка суто математичного методу розв’язання із використанням тільки класичних розділів математики, як: математичний аналіз, лінійна алгебра, дискретна математика та функціональний аналіз є дуже складною, а ймовірно і неможливою задачею. Це в першу чергу спричинено реальною природою досліджуваного сигналу та середовища з якого він походить. Саме через це велику увагу було приділено опису фізичної моделі і відповідно, метод розв’язання повинен використовувати більш наближений до реальних процесів апарат.

Слідуючи змістовній постановці задачі, потрібно знайти методи для рішення чотирьох підзадач, першою з яких є підвищення рівня корисного сигналу в записі. В силу дуже малої амплітуди сигналу, покладатися на пошук форми сигналу не можна, оскільки він може мати значну кількість перешкод, до того ж рівень сигналу на всьому часі запису ніколи не буде однаковим. Як було описано у фізичній моделі, сигнал має певну частотну характеристику, при чому вона не повинна сильно змінюватись із часом, оскільки стан елементів, що спричиняють сигнал також не змінюється. Отже для вирішення цієї підзадачі можна використати

вивчення АЧХ(амплітудно-частотної характеристики)[10] сигналу та вдатися до використання частотних фільтрів[11], які дозволять виділяти тільки корисні частоти і зменшувати значення перешкод.

Наступною підзадачею та ключовим кроком, як видно за математичною постановкою, є визначення частоті коливань механізму b, це значення відкриває шлях до великої кількості подальших кроків. Завдяки відомій множині значень B, цей крок є достатньо простим – використовуючи очищений запис, можна підрахувати кількість пікових значень, попередньо пропустивши його через функцію подібну до gate (рисунок 3.6)[12], що допоможе відсікати майже усі сигнали окрім піків.

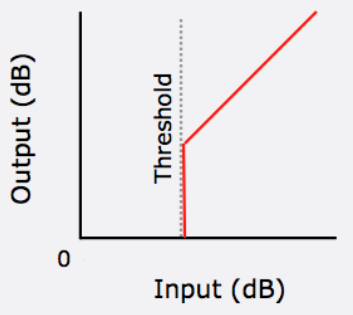

Рисунок 3.6 – Графік функції ґратки (gate)

Використовуючи підраховану кількість та час запису t_g можна достатньо просто знайти найближче значення із множини B. Хоча цей метод не є точним, достатньо велика різниця між значеннями елементів B, дозволяє точно наблизити значення навіть за умови того, що у відфільтрованому записі все ще присутня певна кількість перешкод з великою амплітудою.

Третьою підзадачею є знаходження відрізків, на яких знаходиться корисний сигнал. Цю задачу модна вирішити, використовуючи, знайдені на попередньому етапі піки сигналу та уже відоме значення b. Завдяки відомому еталонному періоду T_b можна відкинути піки, які порушують цей період на певне відхилення $T_{\Delta max}$, при чому це відхилення може бути достатньо малим, що зумовлено самою точністю роботи годинника, використовуючи припущення про екстремуми $f_{r\ maxj}$, можна з високою точністю стверджувати, що залишені піки, є третіми звуками корисного сигналу.

Нарешті останньою і найскладнішою задачею є ідентифікація початку корисного сигналу (першого звука). Як було описано в розділі фізичної моделі, складність спричинена сильною неоднорідністю другого звуку та можливою малою амплітудою першого.

З математичної точки зору можна сказати, що про функцію f_r на частинах відрізків із корисним сигналом між третім звуком та початком сигналу – невідомого нічого, окрім максимальної віддаленості початку сигналу від третього звука, що зумовлена t_{sigmax} та абстрактного розуміння, що на початку сигналу існує певна форма піку першого звука. Пошук першого звука для кожного сигналу із відомими даними є просто неможливим, але цих даних може бути достатньо, якщо підключити апарат теорії ймовірностей та математичної статистики.

Використовуючи поняття випадкової величини[13], можна створити функцію розподілу корисного сигналу за допомогою синтезування сигналу, тобто знаходження середнього значення сигналу у кожний момент часу на відрізку від третього звука до $-t_{sigmax}$, таким чином, синтезувавши розподіл усього сигналу на ньому можна буде знайти певний невеликий відрізок, використовуючи відому абстрактну форму сигналу. Синтез сигналу згладить його піки, але зробить форму більш чіткою, теоретичний вигляд функції синтезованого сигналу можна побачити на рисунку 3.7.

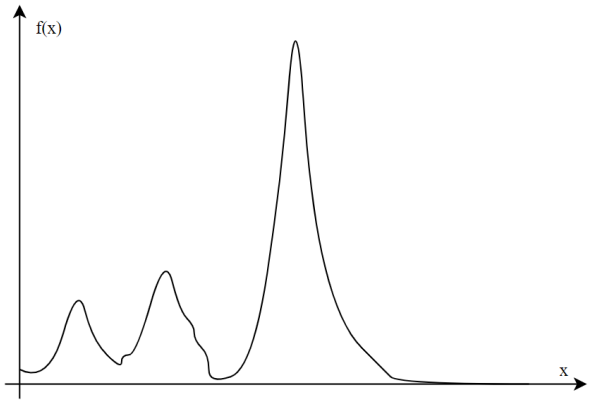

Рисунок 3.7 – Теоретичний вигляд функції синтезованого сигналу

Можна наближено припустити, що цей невеликий відрізок є функцією розподілу випадкової величини піку першого звука, оскільки величина отриманої функції на цьому відрізку буде зумовлена статистичною величиною амплітуди в

момент часу, а отже це значення є рівносильним ймовірності того, що пік першого звука відбувається в конкретний момент часу. Також можна припускати, що при спрямуванні кількості відрізків корисного сигналу до нескінченності, ця випадкова величина набуде розподілу, близького до нормального[13](рисунок 3.8), що притаманно багатьом реальним величинам. Адже дійсно в природі роботи годинникового механізму, початок першого звука може бути відхилений в обидві сторони за рахунок наявних помилки ходу та помилки такту.

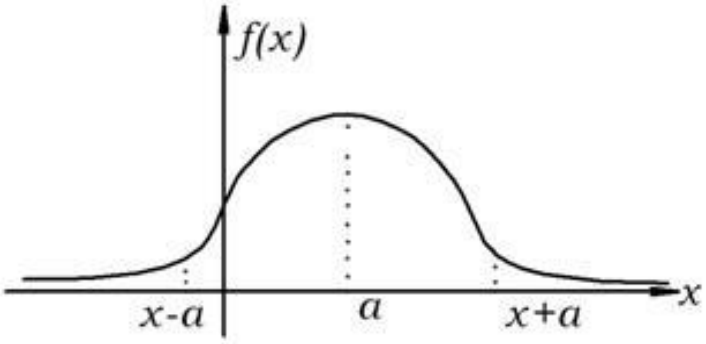

Рисунок 3.8 – Приклад графіку функції нормального розподілу

Знаючи межі розподілу першого звука та розширивши їх певним невеликим значенням для компенсації можливої похибки, можна знайти відносно невеликий проміжок для пошуку піку першого звука у кожному індивідуальному сигналі. Найголовнішим є те, що цей проміжок ніяк не залежить від неоднорідного другого звука.

3.5 Опис методу розв’язання

Метод розв’язання задачі розроблювався на практичному прикладі. Було створено короткий(t_g = 3 секунди) тестовий звукозапис роботи механізму Orient f6922 за допомогою вбудованого мікрофона iPhone 12 mini. Для роботи із тестовим записом також було створено окрему демонстраційну версію додатку, ця версія була розроблена у кінцевий метод розв’язання.

Як зазначалося в розділі 1, однією з основних цілей розробки є створення системи у формі веб-додатку, для цього було обрано використовувати Javascript Web Audio API, який дозволяє працювати зі звуковими файлами у браузері на стороні клієнта.

У першу чергу, значення дискретизованого сигналу було нормалізовано та виведено графік його хвильової форми відрізками по 500 мс для розуміння вигляду вхідних даних. Три перших відрізки показано на рисунку 3.9.

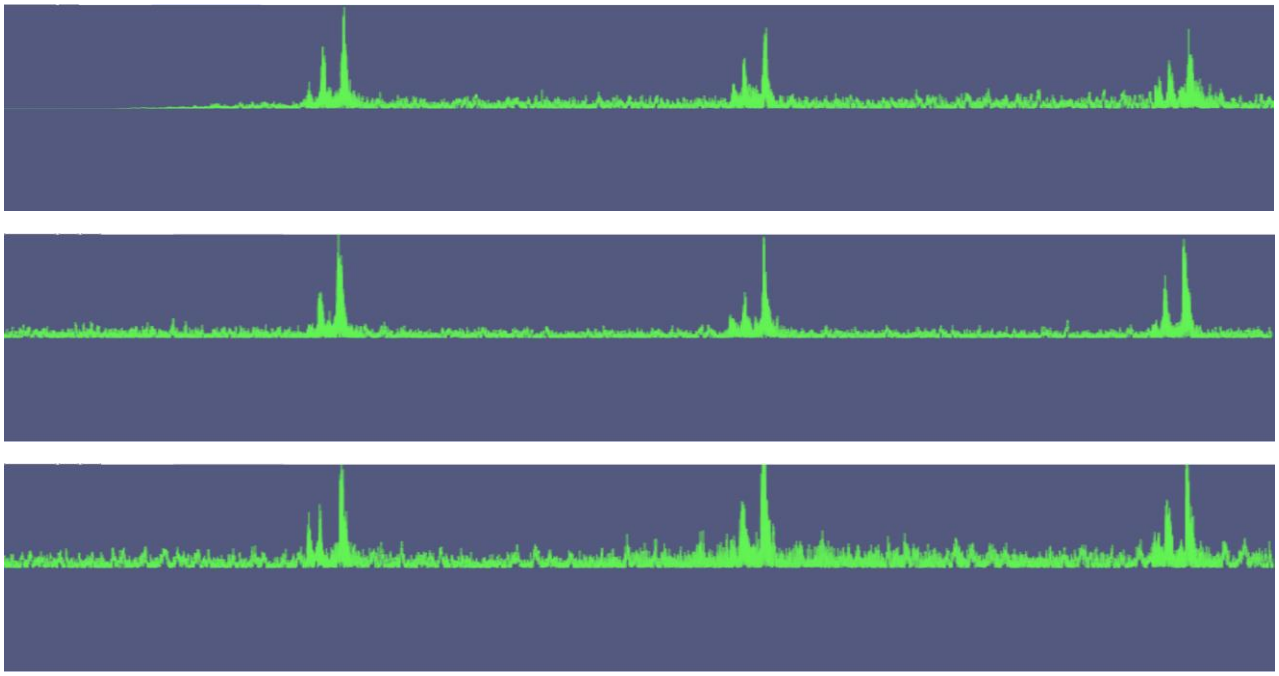

Рисунок 3.9 – Приклад вхідного звукового сигналу

3.5.1 Збільшення рівня корисного сигналу у записі

Як можна побачити, перешкоди є значно сильнішими на деяких відрізках, а розрізнити перший звук навіть візуально майже неможливо. Тому для частотної очистки спочатку було згенеровано графік АЧХ запису, яка по суті є перетворенням Фур'є всього запису, графік зображено на рисунку 3.10, та графік STFT (Short Time Fourier Transform), що є окремими АЧХ для коротких проміжків часу, який зображено на рисунку 3.11.

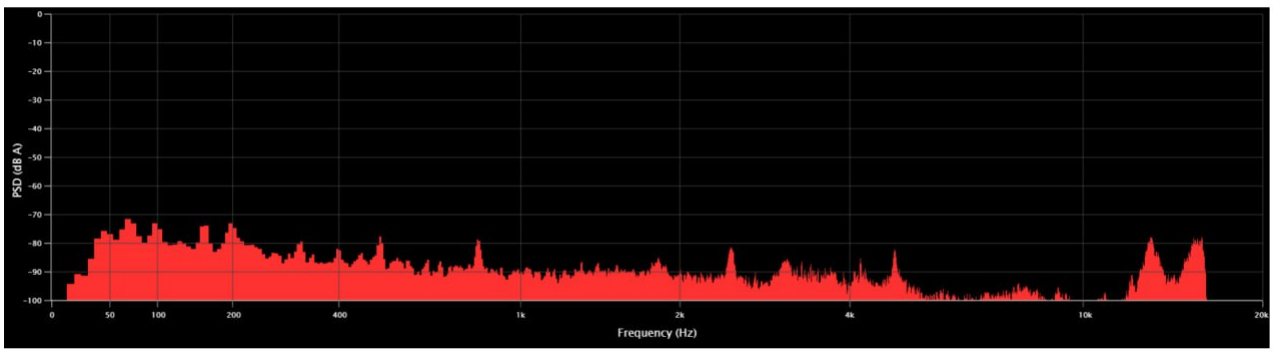

Рисунок 3.10 – АЧХ (Fourier transform) тестового звукозапису

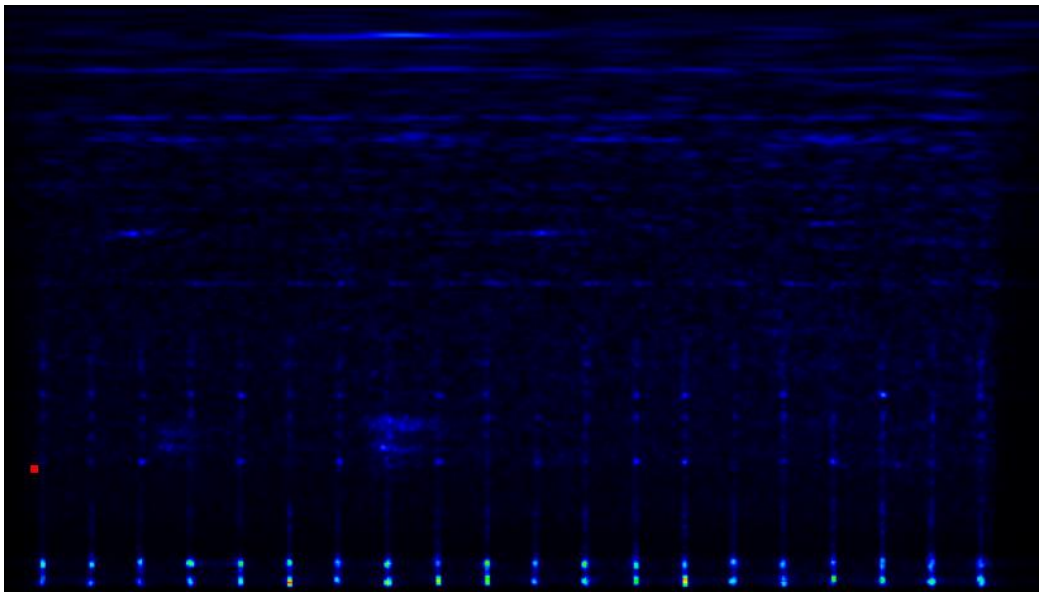

Рисунок 3.11 – STFT тестового звукозапису, точкою відмічено частоту 4750 Hz

Оскільки STFT показує частотну характеристику малих проміжків часу по горизонталі та частоту по висоті, за графіком чітко видно приблизні моменти корисного сигналу. Як і передбачалось у фізичний моделі, звук сигналу є шумоподібним і розподілений по дуже широкому частотному спектру (по висоті), «тіки» починають проглядатись приблизно від 1700 Hz та видні аж до 15000 Hz, також видно, що звук «тіку» має дуже значну високочастотну складову від 10000 до 15000 Hz, те саме проглядається і на АЧХ.

Враховуючи проведений частотний аналіз було вирішено створити комбінований частотний фільтр за допомогою стандартних аудіо вузлів Web Audio API[15], послідовність вузлів зображено на рисунку 3.12.

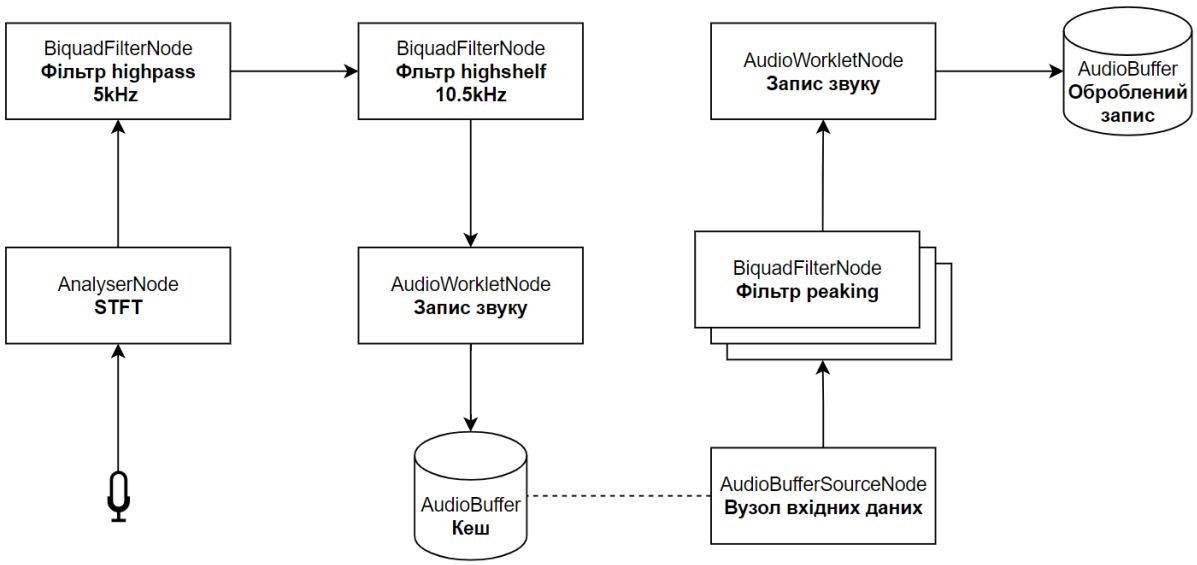

Рисунок 3.12 – Пайплайн вузлів частотної обробки

Для побудови пайплайну було використано чотири типи аудіо вузлів:

- AnalyserNode – вузол який може виконувати перетворення Фур’є за даними які пройшли через нього;

- AudioBufferSourceNode – вузол який може використовуватись як початковий у послідовності обробки та завантажувати у себе будь який закешований звуковий буфер;

- BiquadFilterNode – вузол який призначений для використання стандартних частотних фільтрів, тип фільтру передається як параметр при створенні вузла;

- AudioWorkletNode – вузол який зберігає в собі будь-яку нестандартну логіку, визначену розробником, в даному випадку був написаний вузол для запису даних звуку у буфер.

Першою та основною частиною пайплайну є вирізання полоси частот від 5000 до 10500 Hz, такі значення були вибрані на основі аналізу можливих сторонніх звуків на записі. Наприклад, стандартна розмовна частота людського голосу, чоловічого і жіночого, обмежена 4000 Hz, різні фонові шуми зазвичай мають ще нижчу частоту (як видно на рисунку 3.11), в той же час частоти вище 10000 Hz можуть пересікатись із радіохвилями та певними електронними перешкодами[14]. Таким чином виділивши цю полосу можна з великою ймовірністю можна позбавитись від багатьох шумів та перешкод, в той же час завдяки широкому розподілу корисного сигналу по спектру, буде збережено достатню частину амплітуди корисного сигналу. Вирізання полоси відбувається за допомогою послідовного застосування двох фільтрів:

- спочатку застосовується фільтр highpass, який пропускає усі частоти вище заданої границі, в даному випадку 5000 Hz і майже повністю послаблює нижчі частоти, графік функції фільтра зображено на рисунку 3.13;

- другим застосовується фільтр highshelf, який послаблює частоти, що знаходяться вище заданої границі, в даному випадку 10500 Hz, графік функції фільтра зображено на рисунку 3.14.

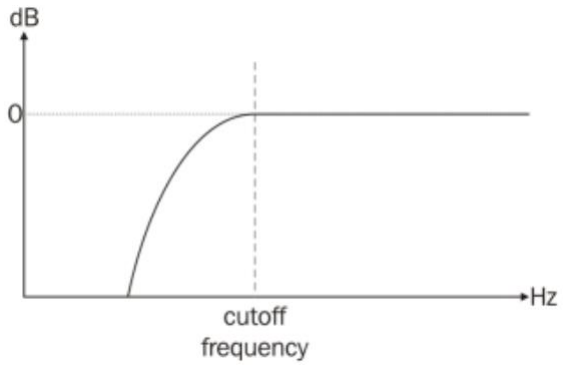

Рисунок 3.13 – Функція highpass

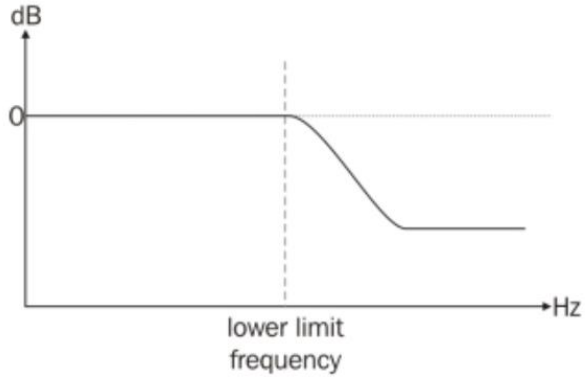

Рисунок 3.14 – Функція highshelf

Паралельно із частотними фільтрами, як видно з рисунку 3.12, також буде записуватись результуючий сигнал та будуватись АЧХ за допомогою перетворення Фур'є, заданий розмір перетворення Фур'є – 2048, це означає, що квант частоти (ціна поділки) готової АЧХ буде дорівнювати $s_r/2048$, у стандартному випадку s_r = 48000 Гц, а отже розмір кванту частоти буде дорівнювати 23.5 Гц.

У другій частині пайплайну застосовується динамічний піковий частотний фільтр. За допомогою АЧХ, побудованої на попередній частині, визначається 8 найбільших піків у вирізаній полосі, групуванням по 2 штуки, із інтервалом в 20 квантів, тобто 470 Гц, таким чином шукаються різні значні гармоніки сигналу, які можуть бути дещо різними для різних механізмів. Далі записаний на попередній частині сигнал використовується як вхід до послідовності фільтрів peaking, які можуть посилювати, або послаблювати задані частоти, та певну полосу навколо них, ширина якої задається значенням Q-фактора[15]. В даному випадку використовується посилення, графік функції фільтра peaking зображено на рисунку 3.15. Це дозволяє збільшити амплітуду певних важливих частот, що робить корисний сигнал трохи більш виразним.

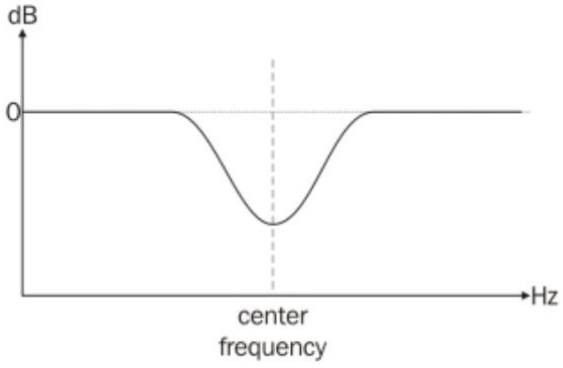

Рисунок 3.15 – Функція peaking (випадок послаблення)

Далі результуючий сигнал знову записується у фінальний буфер, який використовується для подальшої обробки.

Після пропускання початкового запису через побудований фільтр маємо результат зображений на рисунку 3.16. Можна бачити, що амплітуда перешкод та сторонніх шумів значно зменшилась, а перший звук тепер розрізняється візуально у кожному корисному сигналі.

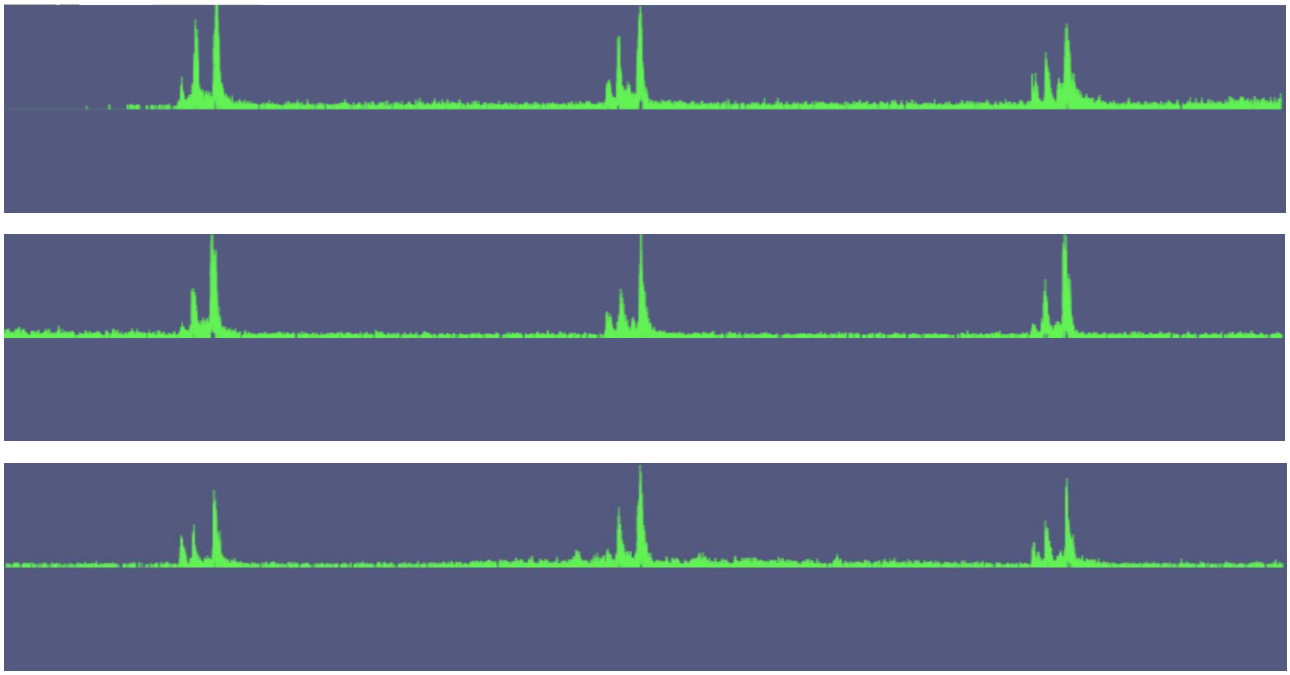

Рисунок 3.16 – Сигнал після застосування частотних фільтрів

3.5.2 Визначення частоти коливань механізму

Для визначення невідомої b достатньо порахувати кількість піків певної величини та знаючи час запису t_g знайти найближче значення із фіксованої множини B.

Для цього в першу чергу запис ділиться на секції певної сталої довжини, це робиться для того щоб уникнути втрачання корисних піків у глобальному контексті, як видно із рисунку 3.16, величина піків третього звуку у сигналі може достатньо сильно відрізнятись, це може бути пов’язано із рівнем шуму в даний момент часу, або автоматичними підлаштуваннями мікрофона та іншими зовнішніми чинниками, саме тому було вирішено ділити запис на секції по 500 мілісекнуд. Цей розмір є одночасно достатньо великим, для того щоб не обробляти

кожний пік окремо та уникнути ситуацій відсутності піків у секції, та достатньо малим, для того щоб уникнути впливу можливої гучного перешкоди на весь запис, якщо така перешкода зустрінеться, то її вплив буде ізольовано в одній секції.

Далі на кожній секції знаходиться значення абсолютного максимуму r_{secmax} і за допомогою функції ґратки виключаються усі значення менші за $r_{secmax}\times 0.7$ (припускається, що після фільтрування навіть за наявності значних перешкод, вони не будуть перевищувати піки корисного сигналу більше ніж на 30%). Наступним кроком підраховується кількість піків у кожній секції з урахуванням можливих випадків, коли пік другого звуку залишився після застосування ґратки. Для цього використовується значення t_{amax}, після знаходження кожного піка, перевіряється що на відстані t_{amax} від нього не існує більших значень, якщо такі є, враховується тільки останнє значення.

Нарешті, підраховується загальна кількість піків, та знаходиться значення частоти коливань механізму, як найближче до фактичного. Знайдені піки можна побачити на рисунку 3.17.

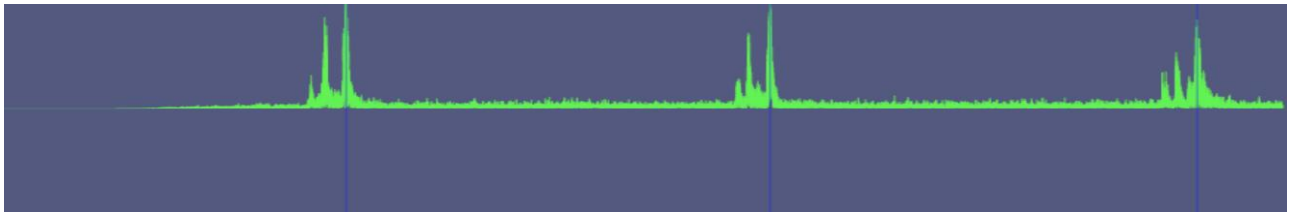

Рисунок 3.17 – Візуалізація знайдених піків

3.5.3 Ідентифікація відрізків корисного сигналу

Наступним етапом є знаходження відрізків корисного сигналу. За рахунок того, що максимальний розмір відрізка відомий із умови задачі, достатньо переконатися, що знайдені піки третіх звуків корисного сигналу дійсно є ними та у тому, що всі піки були знайдені.

В цілому можна сказати, що у сигналі може виникнути 3 основні проблемні ситуації:

- у проміжку між двома корисними сигналами існує значна перешкода, яка була ідентифікована як пік корисного сигналу;

- корисний сигнал розірвано між двома послідовними секціями між другим та третім звуком, і пік ідентифіковано на обох секціях;

- через недостатню амплітуду певного корисного сигналу, його пік було відкинуто при початковому проході.

Загальним підходом для вирішення цих ситуацій буде повторна ідентифікація.

Для того, щоб уникнути першої проблеми, необхідно ідентифікувати хибні піки та виключити їх зі списку. Для цього достатньо знайти перший корисний сигнал, це можна зробити за допомогою послідовного перегляду трьох найближчих піків від поточного піку(припускається, що між двома корисними сигналами не буде більше двох значних перешкод), якщо хоча б один із них знаходиться на еталонному інтервалі від поточного із певним невеликим коефіцієнтом відхилення $e_{int} = 0.04$, можна впевнено казати, що пік не є хибним, після знаходження першого піку, за допомогою того самого еталонного інтервалу та допустимого відхилення e_{int} можна відкинути усі хибні піки.

Можливість вибору настільки малого значення допустимого відхилення зумовлене точністю самого годинникового механізму, наприклад при b = 21600 - T_b = 166.7 мс, а фактичне значення e_{int} = 6.7 мс, або +-3.35 мс, при відхиленні коливань механізму із такою частотою коливань на 3 мілісекунди, відхилення за добу складатиме 26 хвилин, що є майже неможливим для сучасних механізмів.

Для загального вирішення другої проблеми можна зсунути увесь запис вліво таким чином, щоб інтервал між початком запису і першим корисним сигналом дорівнював половині еталонного інтервалу i_{Tb}. Для цього достатньо знайти поточну відстань між першим піком та початком запису, якщо вона перевищує половину інтервалу, відняти від неї різницю, якщо ж вона менша, то обрізати поточну відстань + половину інтервалу (це виключить один пік із запису, що не є критичним).

Нарешті для вирішення третьої проблеми необхідно за допомогою значення еталонного інтервалу визначити місця у яких повинні бути піки сигналу, але не біли ідентифіковані в силу замалої амплітуди відносно своєї секції. Для виправлення

цього достатньо використати ту саму функцію, яка використовується на етапі визначення частоти коливань, але цього разу на малому відрізку, який повинен включати тільки один пік. Таким чином, навіть якщо неідентифікований пік є у декілька разів меншим за сусідні, при пошуку у меншому відрізку, його буде ідентифіковано. На рисунку 3.18 можна побачити першу секцію після виконання цих кроків.

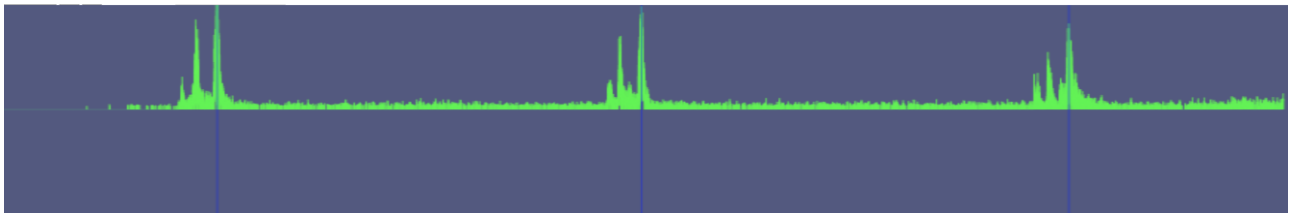

Рисунок 3.18 – Піки після уточнювальної ідентифікації

3.5.4 Ідентифікація початку сигналу

Останнім та найважливішим кроком вирішення є ідентифікація початку корисного сигналу, тобто перших звуків, які необхідні для проведення кінцевих обчислень.

Перед початком пошуку за допомогою теорії ймовірностей та математичної статистики визначається розподіл піку першого звуку. Для того щоб зробити це введемо випадкову величину P_1, значенням якої є відстань від піку третього звуку, який відомий за результатом попередніх кроків, до піку першого звуку. Графічне представлення величини наведено на рисунку 3.19.

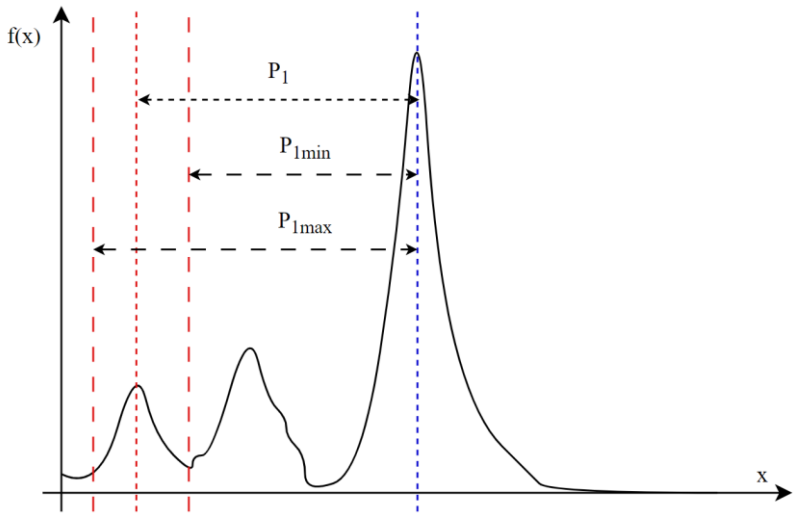

Рисунок 3.19 – Графічне представлення випадкової величини P_1

Очевидно, що значення P_1 будуть знаходитись у певних межах P_{1min}, P_{1max}. Для знаходження цих меж побудуємо статистичну форму проміжку корисного сигналу, іншими словами синтезуємо сигнал. Спочатку створимо відрізки однакового розміру, використовуючи відомі точки піків третіх звуків як опорні, за умовою відомо, що максимальна тривалість усіх трьох звуків t_{sigmax} = 30 мс, додамо відрізок на 35 мілісекунд вліво від опорної точки та на 15 мілісекунд вправо, включаючи затухання третього звука, таким чином отримаємо відрізки із корисним сигналом. З точки зору теорії ймовірності можна розглянути кожний такий відрізок-екземпляр як повторюваний експеримент, який представляє собою вектор значень амплітуди. Якщо поелементно скласти усі значення експериментів та поділити на їх кількість, отримаємо значення, що приблизно відображає наскільки часто пік звуку опинявся у конкретному елементі, звичайно також має вплив величина самих значень у кожному експерименті, як можна бачити найменші точки першого звуку в одних сигналах є більшими за найбільші в інших, але цей вплив зменшується зі збільшенням кількості експериментів n_{ex}, оскільки самі значення усереднюються. Таким чином отримуємо, що значення по осі абсцис є значенням випадкової величини P_1, а значення по осі ординат відображають псевдоймовірність, яка наближується до звичайної ймовірністю із прямуванням кількості експериментів до нескінченності $n_{ex}\to\infty$. Таким чином отримуємо функцію розподілу випадкової величини. Такий синтезований сигнал на прикладі тестового запису, n_{ex}=18 можна побачити на рисунку 3.20.

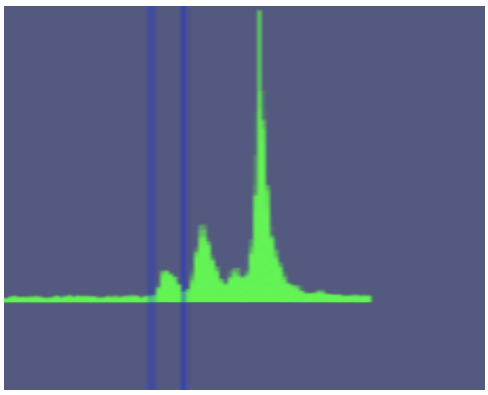

Рисунок 3.20 – Приклад синтезованого сигналу із визначеними межами P_1

Як видно навіть із дуже малою кількістю експериментів, розподіл випадкової величини уже нагадує нормальний, як було припущено раніше.

Далі необхідно знайти межі P_1 за побудованим розподілом, це можна зробити за допомогою знання форми сигналу, а конкретно того, що між першим та другим звуками буде або провал в амплітуді, або вони зіллються у сходинку через неоднорідність другого звуку. Таким чином шукаємо P_{1max} як першу точку від початку відрізка, в якій іде значний зріст функції, а P_{1min} як початок зросту після затухання, що знаходиться після точки P_{1max}, або як значний зріст без затухання у випадку утворення сходинки. Обидві точки знаходяться у реалізації, за допомогою порівняння поточного значення та середнього значення на попередньому відрізку, це відношення повинне перевищувати заданий коефіцієнт $k_{trig} = 2.5$ для P_{1max} та $k_{trig} = 2$ для $P_{1min.}$ За такого методу пошуку меж знайдені точки будуть лежати посеред відрізків зростання, тобто обидві точки будуть зміщені вправо, тому від обох віднімаємо певне невелике значення для компенсації.

Дуже великою перевагою такого методу є те, що завдяки переводу кожного елемента до середнього арифметичного, усі випадкові шуми та перешкоди дуже швидко стираються, шанс багатократного повторення таких перешкод в одному місці майже нульовий, відповідно знайдені зростання можуть бути пов’язані тільки зі звуками корисного сигналу.

Останнім кроком для вирішення задачі, знаючи межі випадкової величини P_1 та точки піків третього звуку в кожному екземплярі корисного сигналу, достатньо провести пошук у коротких відрізках $[P_{1maxj}.. P_{1minj}]$ першого значення яке складає не менше 80% від локального максимума відрізка (така логіка пов’язана із тим, що іноді пік першого звуку не є його початком). Знайдені таким чином початки корисного сигналу можна побачити на рисунку 3.21.

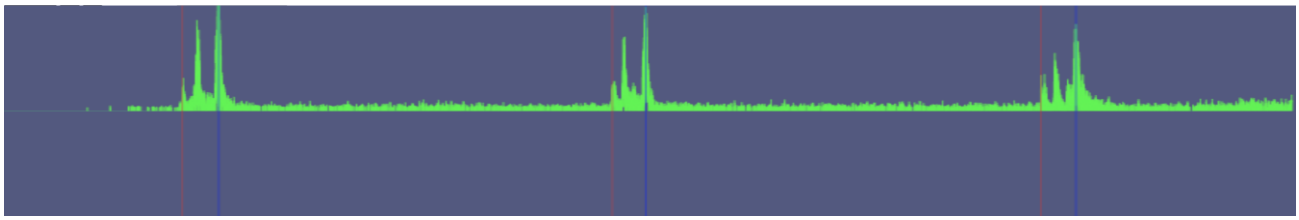

Рисунок 3.21 – Знайдені початки корисних сигналів

Розрахунок інтегральних характеристик просто виконується за допомогою формул із математичної постановки задачі.

Висновок до розділу

У розділі було описано фізичну модель процесу, оскільки вона має сильний вплив на вирішення задачі. Було чітко поставлено змістовну задачу та сформовано математичну постановку задачі. Було обґрунтовано використання математичного апарату. В кінці було детально описано комбінований метод вирішення задачі, який включає кроки пов’язані із фізичною математикою, елементарними прийомами алгебри та математичного аналізу, а також застосування теорії ймовірностей та математичної статистики.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для створення продукту було задіяно велику кількість технічних засобів, які можна поділити на чотири основних категорії: інфраструктура, бекенд розробка, фронтенд розробка та CI/CD(Continuous Integration/Continuous Delivery). Нижче наведений список усіх технічних засобів за категоріями.

Засоби бекенд розробки:

- Node.js;
- Fastify;
- Pino.

Засоби фронтенд розробки:

- React;
- Javascript Web Audio API;
- Javascript Media API;
- Javascript Canvas API;
- SASS;
- Fontawesome.

Засоби інфраструктури:

- Google Cloud Platform (GCP);
- Google Cloud Run;
- Google Firestore;
- Google Cloud Storage;
- Google Oauth API;
- Terraform.

Засоби CI/CD:

- GitHub;
- Docker;
- Google Cloud Build;
- Google Artifact Registry.

Для створення коду використовувались мови Javascript та Typescript.

Далі наведені короткі описи технічних засобів із прикладами їх використання в продукті.

4.1.1 Засоби бекенд розробки

Node.js

Node.js є крос-платформеним середовищем для роботи із мовою Javascript, зокрема у бекенд розробці, також він представляє спеціальні розширені API для Javacript, які дозволяють наприклад взаємодіяти із файловою системою, або мережевими протоколами та портами машини, зокрема HTTP, це досягається завдяки додаванню декількох ключових бібліотек написаних на мові C++.

Node.js є також асинхронним event-driven середовищем, що робить його прекрасним варіантом для створення серверних додатків з точки зору механізмів обробки вхідних мережевих запитів. [16]

Fastify

Fastify являється одним із найшвидших HTTP фреймворків для Node.js. Окрім своєї продуктивності Fastify має розширений та чітко визначений життєвий цикл HTTP запиту та відповіді, а також можливість влаштування власної логіки до конкретних етапів циклу за допомогою хуків (Hooks). Наприклад, у бекенд додатку системи в цій роботі використовується хук onRequest[17], який запускається після валідації запиту, за допомогою нього реалізований механізм аутентифікації(рисунок 4.1), кожний запит, що має HTTP методи POST, PATCH, DELETE, перевіряються на наявність заголовка Authorization, та валідацію токена, що відправляється у ньому.

```
18
19    server.addHook('onRequest', authHook);
20
```

Рисунок 4.1 – Використання хука onRequest для аутентифікації

Також Fastify має вбудовану підтримку JSON-схем[18], що використовуються для будь-яких валідацій вмісту різних частини HTTP запитів(тіло, заголовки, url), при чому завдяки схемам ці валідації описуються

повністю декларативно. Fastify автоматично використовує схеми для валідації вхідних запитів і у випадку помилки, відправляє відповідну HTTP відповідь. Такий підхід дозволяє сильно зменшити повторне написання стандартного коду. В даній роботі схеми Fastify були використані для абсолютно всіх HTTP ендпойнтів бекенд додатку.

Ще однією великою перевагою Fastify є його модульна система, яка називається Fastify Plugins[19]. Ця система дозволяє створювати новий функціонал для фреймворка та піключати його простим стандартним та безпечним шляхом до ядра Fastify. Ця перевага дозволила всьому ком'юніті розробити величезну кількість розширень до логіки фреймворка, які можуть підключатись одним рядком коду. Наприклад, у цій роботі було використану плагін fastify-swagger, який автоматично генерує документацію за стандартом OpenApi до всіх ендпойнтів додатку та представляє її візуально на HTML сторінці, рисунок 4.2.

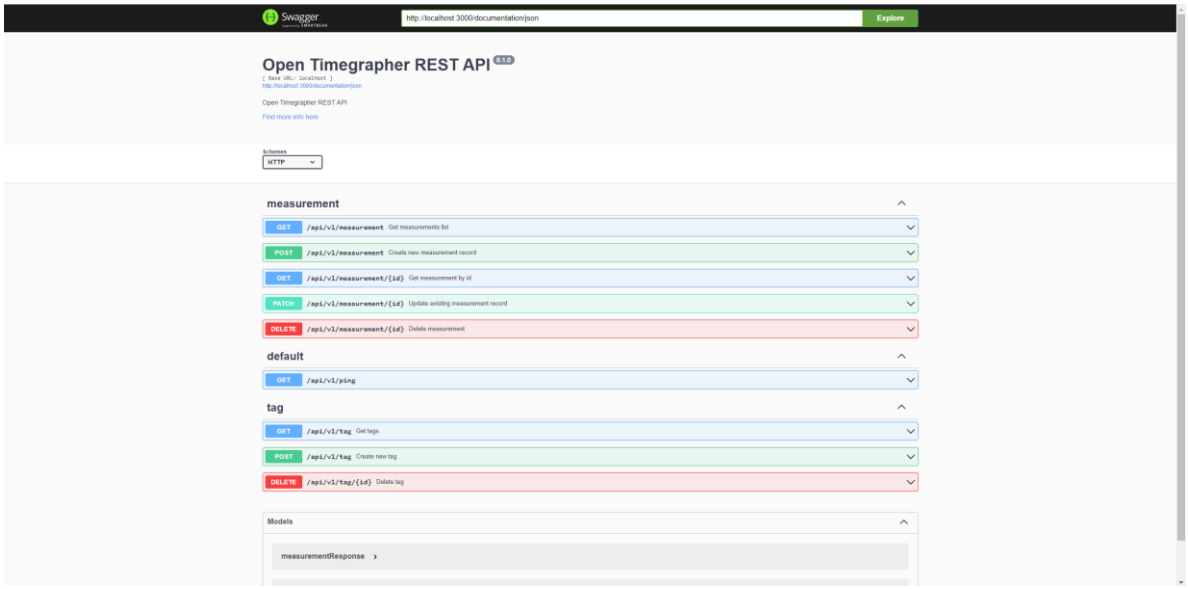

Рисунок 4.2 – Автоматично згенерована документація fastify-swagger

Pino

Pino – це найшвидший логер для Node.js додатків. Він використовує власні механізми сериалізації об'єктів та виводу, що дозволяє досягати найвищої продуктивності. Pino представляє як стандартні можливості логування, такі як різні рівні логування та вибір рівня, так і розширені можливості, такі як вибір перенаправлення вихідного потоку до будь-яких виходів та створення власних

функцій форматування. В даній роботі Pino використовується для логів HTTP серверу, та логів додатку.[20]

4.1.2 Засоби фронтенд розробки

React

React є одним із найпопулярніших фронтенд фреймворків односторінкових додатків і напевно найбільш універсальним. Основними особливостями React є JSX, використання реактивної моделі та React hooks.[21]

React використовує реактивну модель для своєї роботи за допомогою збереження та відстежування стану додатку, або його окремих компонентів. Зміна стану миттєво перетворюється на зміну вигляду сторінки, компонентів або призводить до певних дій. Реактивна модель добре підходить для створення графічних інтерфейсів, оскільки їх логіка зазвичай є досить складною, React дозволяє описувати тільки стан елементів графічного інтерфейсу в залежності від стану системи, а також представляє можливості для збереження та зміни стану.

JSX є унікальною концепцією синтаксису, що дозволяє використовувати розмітку HTML тегів в ролі виразів і змінних мови Javascript, це дозволяє змішувати логіку розмітки та логіку її відображення в єдиному синтаксисі, що є дуже корисною можливістю. (рисунок 4.3).

```
return (
  <button
    className={inverted ? styles.ButtonInverted : styles.Button}
    style={{ fontSize: `${size}em` }}
    onClick={onClick ? (event) => onClick(event) : () => false}
    disabled={disabled}
  >
    {children}
    {icon ? (
      <>
        <small className={styles.dateWindow}>
          <span className={styles.date}>{new Date().getDate()}</span>
          <FontAwesomeIcon icon={icon} />
        </small>
        <Dial />
      </>
    ) : (
      ''
    )}
  </button>
);
```

Рисунок 4.3 – Приклад застосування JSX

Javascript Web Audio API

Нативні інструменти, що підтримується браузерним Javascript для роботи із аудіо даними. Web Audio API містить стандартні інтерфейси для опису структури аудіо файлів та потоків, а також стандартні класи, що дозволяють обробляти аудіо дані.[22]

Ключовим класом є AudioContext, який представляє граф аудіо обробки який складається із вузлів, що представлені екземплярами іншого основного класу AudioNode. AudioNode має велику кількість дочірніх класів, що реалізують конкретну логіку обробки звуку, наприклад AudioBufferSourceNode є вузлом, що може завантажувати у себе аудіо буфер та програвати цей буфер до інших вузлів графу. AnalyserNode дозволяє застосовувати віконне перетворення Фур'є, пропускаючи через себе аудіо дані, тобто отримувати амплітудно-частотну характеристику із хвильової часової форми звукового сигналу. BiquadFilterNode є вузлом, що дозволяє застосовувати базові хвильові фільтри, такі як highpass та lowpass до аудіо даних. Нарешті також існує вузол AudioWorkletNode, який дозволяє створювати власну логіку обробки аудіо даних, та використовувати її як стандартний вузол у графі обробки. Усі вищезгадані вузли були використані в даній роботі, а детальний граф обробки звуку представлено на рисунку 3.12.

Javascript Media API

Ще одна нативна бібліотека браузерного Javascript[23], яка дозволяє отримувати доступ та взаємодіяти із вхідними та вихідними медіа пристроями операційної системи, такими як, дисплей, камера, динаміки, навушники та мікрофон. API представляє набір стандартних інтерфейсів, що спрощує роботу із даними, одним із них інтерфейс потоку медіа даних MediaStream, який можна інтегрувати із Web Audio API за допомогою вузла MediaStreamAudioSourceNode. В даній роботі це API використовувалось для отримання потоку аудіо даних із вбудованого мікрофону мобільних пристроїв, таких як ноутбуки та смартфони.

Javascript Canvas API

Є нативним API браузерного Javascript для створення графіки на сторінці. API представляє усі базові можливості для відмальовування графіки, такі як

стандартні геометричні фігури, та лінії, а також більш складні інструменти, як графічні шейдери. API інтегрується у сторінку за допомогою спеціального тегу <canvas>. В Даній роботі API використовувалось здебільшого для відображення хвильової форми звукового сигналу, як статичного, так і анімованого, що було використану у програвачі записів на одній із сторінок додатку.[24]

SASS

SASS являється препроцесором для CSS. Він розширює стандартний синтаксис CSS такими можливостями як імпорт змінних, створення вкладених стилів та використання наслідування для класів стилів.

Fontawesome

Відкрита бібліотека векторних іконок, використовується для створення графічних інтерфейсів фронтенд веб-додатків. Окрім самих зображень іконок також надає стандартні API для їх інтеграції в розмітку сторінки, зокрема Fontawesome має бібліотеку React компонентів, що спрощує використання у React додатках.

4.1.3 Засоби інфраструктури

Google Cloud Platform

GCP – це хмарна платформа компанії Google, яка зокрема призначена для розгортання будь-яких типів додатків, у тому числі – веб-додатків. Є однією із найбільш використовуваних хмарних платформ та відрізняється своєю добре продуманою структурою. Надає можливості використання як базових хмарних апаратних ресурсів так і різних високорівневих продуктів за моделями PaaS, SaaS, FaaS та Serverless. Використання хмарних ресурсів забезпечує велику надійність додатку, а також, часто, можливість використання більш простих інструментів для розгортання додатку.

Великою перевагою GCP є також щедрі безкоштовні ліміти на використання продуктів, їх використання несе певну специфіку та складності, але загалом надає великі можливості. Вся, достатньо нетривіальна інфраструктура та частина CI/CD засобів продукту даної роботи була побудована на GCP повністю безкоштовно і

може навіть оперувати у рамках безкоштовних лімітів під певними невеликими навантаженнями.

Google Cloud Run

Cloud Run є одним із продуктів GCP, він являється повністю автоматичною serverless обчислювальною платформою. Це означає, що кінцевий користувач платформи майже не повинен перейматись станом розгорнутого додатку, так само як і регулюванням навантаження на нього, оскільки внутрішні механізми Cloud Run будуть самостійно стежити за тим, щоб додаток завжди знаходився в робочому стані й у виникненні критичної помилки та відключенні додатку, автоматично запустить його знову, також сам Cloud Run буде повністю відповідати за масштабування додатку, наприклад при відсутності навантаження він може повністю перевести додаток в режим очікування, що дозволяє економити ресурси, а при збільшенні навантаження автоматично запустить додаток та продовжить автоматичне горизонтальне масштабування стільки, скільки це буде необхідним, з урахуванням лімітів налаштованих користувачем.[25]

Як і всі сучасні обчислювальні платформи, Cloud Run працює із контейнеризованими системами, тобто для того, щоб розгорнути додаток у Cloud Run потрібно мати його побудований image (образ).

Cloud Run є наступним рівнем абстракції над ресурсами після Kubernetes і насправді використовує велику частину функціоналу Kubernetes «під капотом», але це майже непомітно для кінцевого користувача.

Загалом Cloud Run є однією із найсучасніших, надійно протестованих та найзручніших обчислювальних платформ. В продукті цієї роботи Cloud Run використовується як обчислювальна платформа для розгортання як бекенд так і фронтенд додатків.

Google Firestore

Firestore також є одним із продуктів GCP – serverless NoSQL документна база даних, що підтримує JSON формат документів. Firestore є представником serverless продукту, як і Cloud Run, це означає, що кінцевий користувач не має прямого доступу та уявлення про апаратні ресурси, використовувані базою даних. Firestore

використовує інший продукт – Google App Engine, який і є serverless обчислювальною платформою для своєї роботи.[26]

Firestore чітко направлений на швидкодію та продуктивність запитів, він автоматично створює індекси для кожного поля кожного документа окремо, для того щоб дуже швидко виконувати базові запити. Проте через цю особливість, Firestore має велику кількість обмежень із точки зору гнучкості запитів, у порівнянні із SQL базами даних, або навіть іншими NoSQL, як MongoDB, його запити є досить скромними. Також із цієї причини, кожний складний запит (запит що використовує декілька операторів на декількох полях) потребує створення окремого індексу саме під цей запит. Також індексування структур даних таких як масиви є обмеженим і часто потребує оригінального підходу до самої структури даних для забезпечення функціоналу. Враховуючи вище сказане, Firestore не є абсолютно універсальним, проте для систем, що потребують невеликої кількість запитів та мають ненормалізовані записи, що включають наприклад масиви та інші об’єкти Firestore є ідеальним варіантом в силу своєї швидкості та serverless природі, що забезпечую дуже хорошу надійність.

Firestore використовується як основна база даних продукту цієї роботи, разом із певною кількістю окремих індексів для забезпечення декількох найскладніших запитів.

Google Cloud Storage

Cloud Storage є одним із найперших та найосновніших продуктів GCP та являється простим хмарним файловим сховищем. Cloud Storage використовується як допоміжний інфраструктурний елемент для багатьох продуктів GCP, зокрема для Firestore. В продукті даної роботи Cloud Storage використовується для збереження стану системи для інструменту Terraform, цей стан являє собою простий текстовий файл, що робить доречним використання Cloud Storage.

Google Oauth API

Одне із багатьох публічно доступних API від компанії Google, воно забезпечує можливості використання Google в ролі зовнішнього провайдера персоналій (identity provider)[27]. Іншими словами словами, замість створення та

збереження власних користувачів, API дозволяє використовувати стандартні облікові записи Google для ідентифікації користувачів в системі та отримання токена доступу. Для використання API додатком необхідно створити проект у GCP та налаштувати свій додаток на відповідний сторінці. API також надає функціонал для верифікації токенів доступу, надісланих користувачами. В цілому це є ідеальним варіантом для додатків, які не мають задачі створення облікових записів користувачів, а зацікавлені тільки в ідентифікації їх особистості (тобто аутентифікації), що є випадком для продукту даної роботи – Google Oauth API використовується у фронтенд додатку та у бекенд додатку для верифікації HTTP запитів.

Terraform

Terraform від компанії HashiCorp є крос-провайдерним IaS(Infrastructure as Code – Інфраструктура як Код) інструментом. Підхід IaS призначений для того, щоб описувати бажаний стан інфраструктури системи та її конфігурації у вигляді декларативного коду, та роботи із інфраструктурою як із кодом з точки зору його написання, версіонування, злиття та інших операцій.[28]

Terraform надає свій власний декларативний синтаксис для опису інфраструктурних ресурсів, що є змішаним форматом JSON та YAML(рисунок 4.4).

```
resource "google_app_engine_application" "firestore-database" {
  project = var.project_id
  location_id = var.region
  database_type = "CLOUD_FIRESTORE"
}
```

Рисунок 4.4 – Приклад коду Terraform для створення бази даних Firestore

Terraform оперує за допомогою пакетів ресурсів, що називаються провайдерами, кожний провайдер призначений для конкретної інфраструктурної платформи, він має велику кількість готових провайдерів, таких як GCP, Azure, AWS і багато інших і тому є крос-провайдерним. Кожний провайдер одночасно є готовим клієнтом API інфраструктурної платформи, для якої він побудований, наприклад усі перераховані вище популярні хмарні інфраструктурні платформи мають REST API, для створення та використання абсолютно всіх типів ресурсів, ці API використовуються для внутрішньої взаємодії самими платформами. Таким

чином Terraform бере декларативний код користувача як основу для запитів до внутрішніх API платформ, та виконує ці запити до API невидимо для користувача.

Іншою важливою частиною є стан, Terraform підтримую стан всіх створених ресурсів у спеціальному текстовому файлі, для того щоб бути в змозі визначити, які ресурси слід видаляти, які створювати та оновлювати, тому стан є критичним елементом для оцінки конфігурації. За замовчуванням файл стану зберігається у локальній файлові системі користувача, але для забезпечення надійного зберігання стану, та уникнення конфліктів локальних станів декількох користувачів, Terraform може використовувати певні продукти самої платформи для збереження файлу стану, зазвичай це продукти файлового сховища, а у випадку GCP – Cloud Storage.

Коротшими словами, Terraform дозволяє писати декларативний код для створення інфраструктури в конкретній платформі, за допомогою підтримання внутрішнього стану, при чому створення інфраструктурних ресурсів відбувається непомітно для користувача за допомогою нативних API платформи.

Terraform використовується в даній роботі для автоматизації створення всієї інфраструктури системи, включаючи Cloud Run, Firestore та інші ресурси.

4.1.4 Засоби CI/CD

GitHub

GitHub є веб-провайдером системи контролю версій Git та віддалених файлових репозиторіїв для коду. GitHub також має велику кількість інтеграцій із різними інструментами CI/CD та слугує стартовою точкою для будь-яких CI/CD пайплайнів, в тому числі GitHub має можливість інтеграції із Google Cloud Build, що є основним CI/CD продуктом GCP. Використання віддалених репозиторіїв GitHub також дозволяє використовувати операційний підхід GitOps, який застосовується у цій роботі. Для продукту даної роботи GitHub використовується для збереження коду як додатків, так і інфраструктурного коду Terraform.

Docker

Docker є найпопулярнішим інструментом для створення контейнеризованих образів додатків. Образи контейнерів будуються пошарово, а кожний шар

визначається директивою у конфігураційному текстовому файлі Dockerfile. Docker використовується для контейнеризації фронтенд та бекенд додатків продукту даної роботи.

Google Artifact Registry

Artifact Registry є продуктом GCP та представляє собою репозиторій сховище для різних типів артефактів (результатів роботи CI/CD пайплайнів), в тому числі одним із підтримуваних типів артефактів є образи Docker[29]. Великою перевагою Artifact Registry є його фізична близькість до обчислювальних платформ, таких як Cloud Run, що використовують образи для запуску контейнерів. Це дозволяє завантажувати образи за коротший час, а отже швидше розгортати додатки. Також Artifact Registry має вбудовану перевірку образів на наявність вразливостей всіх рівнів.

У продукті даної роботи Artifact Registry використовується для збереження образів фронтенд та бекенд додатків.

Google Cloud Build

Cloud Build є основним CI/CD продуктом GCP, він являється дуже потужним інструментом завдяки своїй особливості виконання пайплайнів CI/CD – для виконання розгортається окрема віртуальна машина, образ якої може визначатись користувачем, як і інструменти для виконання кроків пайплайну, це дозволяє запускати абсолютно різні типи пайплайнів у Cloud Build. Самі пайплайни Cloud Build описуються за допомогою текстових файлів формату YAML.[30]

В даній роботі Cloud Build інтегрується із GitHub репозиторіями додатків та Terraform коду і відповідно запускає два типи пайплайнів:

- створення образу додатків за допомогою Docker та їх розгортання у Cloud Run;
- запуск команд Terraform для застосування змін в інфраструктурі.

4.2 Вимоги до технічного забезпечення

Вимоги до програмного забезпечення системи:

- ОС Windows, Linux, MacOS;

- підтримка контейнирезованих систем;
- Docker.

Вимоги до апаратного забезпечення системи:

- об’єм оперативної пам’яті не менше 8 Gb;
- тактова частота процесора не менше 1 GHz;
- мережевий адаптер із доступом до інтернету.

Вимоги до програмного забезпечення клієнта:

- встановлений браузер із підтримкою мови Javascript стандарту не нижче EcmaScript 2016.

Вимоги до апаратного забезпечення клієнта:

- об’єм оперативної пам’яті не менше 1 Gb;
- справний якісний вбудований або зовнішній мікрофон.

4.3 Архітектура програмного забезпечення

Для розробки продукту цієї роботи було застосовано дві основних архітектури: мікросервісна архітектура та SPA(Single Page Application) – архітектура односторінкового додатку, також був використаний набір архітектурних правил для розробки бекенд API під назвою REST.

Мікросервісна архітектура полягає у розділенні додатку на окремі повністю ізольовані частини, починаючи із розробки та коду, закінчуючи розгортанням, ці частини називаються мікросервісами. Ізоляція мікросервісів проводиться на усіх рівнях життєвого циклу, таких як:

- розробка коду;
- тестування;
- розгортання;
- масштабування;
- підтримка.

Таким чином кожний мікросервіс представляє собою повністю окрему сутність, часто навіть незалежну від інших. Схематичне зображення мікросервісної архітектури можна побачити на рисунку 4.5.[31]

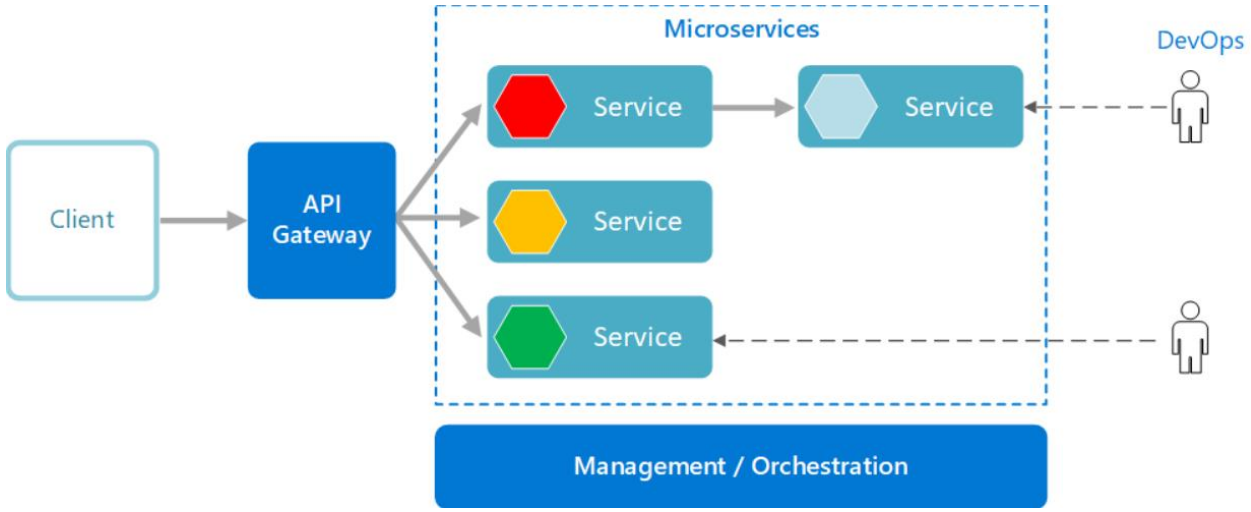

Рисунок 4.5 – Мікросервісна архітектура

Звичайно, в цієї архітектури є і недоліки – ізоляція несе за собою ускладнення інфраструктури та збільшення кількості конфігурації для розгортання продукту.

Через це вибір мікросервісної архітектури повинен бути зваженим. Зазвичай основними індикаторами доречності розділення продукту на мікросервіси є:

- достатньо велика різниця у навантаженні на певні частини системи, де одна частина може майже не працювати, а інша буде знаходитись під великим навантаженням;
- велика відмінність у необхідному для роботи середовищі;
- відмінність у частоті релізів та розгортань частин системи;
- чітке логічне розмежування частин системи.

Для продукту даної роботи особливо чіткими є другий та четвертий індикатори, оскільки двома частинами системи є бекенд та фронтенд додатки, при цьому фронтенд додаток потребує елементарного статичного серверу для своєї роботи, в той час як бекенд потребує повноцінного середовища із Node.js. Третій індикатор також може мати місце, оскільки основний потік сценаріїв використання продукту майже не використовую бекенд додаток.

Наступною архітектурою є односторінковий додаток (SPA). Головним принципом цієї архітектури є робота додатку виключно на стороні клієнта. В односторінковому додатку весь сайт збирається в один або декілька мініфікованих файлів, що називаються бандлами, ці банлди одноразово завантажуються на

сторону клієнта із сервера і містять в собі всю логіку для рендерингу та маршрутизації сторінок. З технічної точки зору, односторінковий додаток використовує тільки один HTML документ(тобто реальну сторінку), а тому завантаження усіх ресурсів як стилі та скрипти відбувається тільки один раз, а перехід на інші сторінки, насправді є відображенням інших компонентів у тому самому HTML документі.

Це дозволяє фронтенд додатку працювати 100% часу та відповідати на дії користувача навіть при завантаженні певних даних із сервера, оскільки вся логіка для відображення уже знаходиться на стороні клієнта і є необхідність у завантаженні тільки корисних даних, а не розмітки. У контрасті більш традиційний багатосторінковий додаток(MPA – Multi Page Application) завантажує новий HTML документ із потенційно новими ресурсами із сервера для кожної сторінки і в цей час фронтенд додаток стає повністю заблокованим. Порівняння циклу роботи односторінкового та багатосторінкового додатків можна побачити на рисунку 4.6.

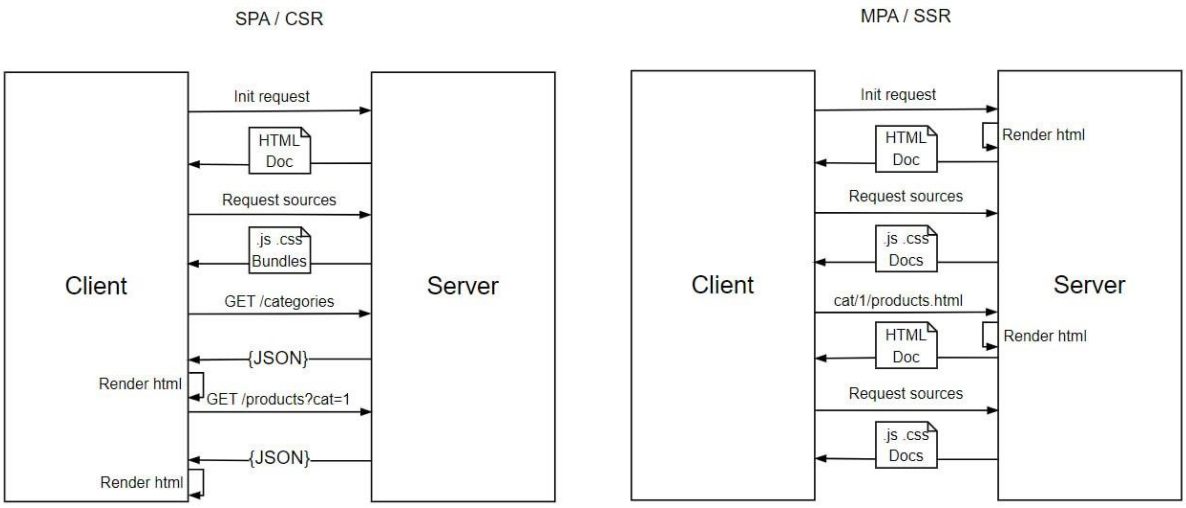

Рисунок 4.6 – Порівняння роботи SPA та MPA

Як можна бачити, завантаження розмітки та стилів відбувається одноразово, а подальші запити до сервера відбуваються у набагато більш стислому форматі JSON. А за відображення відповідає клієнт (CSR – Client Side Rendering).

Нарешті останньою частиною є REST, що представляє собою набір архітектурних правил та рекомендацій для створення бекенд HTTP API. Майже всі сучасні API побудовані за принципами REST, тому це є майже стандартом індустрії

на сьогоднішній день. Причиною популярності REST є його зрозумілість та передбачуваність, це забезпечується такими правилами як:

- тіло HTTP запитів та відповідей завжди повинно мати формат JSON;
- всі ендпойнти API повинні мати властивість ідемпотентності;
- HTTP методи дозволені для використання POST, GET, PUT, PATCH, DELETE, при чому всі вони мають чітку відповідність до логічних операцій над сутностями CRUD (Create, Read, Update, Delete);
- система повинна використовувати клієнт-серверну архітектуру.

Та багато інших зручних правил. Також особливим є те, що велика частина правил є рекомендаціями, тобто їх виконання не вимагається насильно, це в свою чергу дозволяє більш просто розробляти API та мати більшу гнучкість у нестандартних випадках.[32]

4.3.1 Діаграма компонентів програмного забезпечення

Діаграма архітектурних компонентів подана у графічному матеріалі. Система має наступну структуру компонентів:

- фронтенд додаток на мові Javascript:
    - React компоненти – відповідають за відображення графічного інтерфейсу користувача;
    - класи бізнес-логіки для роботи зі звуковим сигналом;
    - класи клієнтів для бекенд API;
    - функціональні модулі із математичними та графічними утилітами.
- бекенд додаток на платформі Node.js:
    - HTTP сервер Fastify;
    - класи HTTP контролерів;
    - класи сервісів сутностей;
    - клас клієнт бази даних;
    - функціональний модуль із утилітами.
- база даних Firestore:

- колекція для зберігання вимірювань;
- колекція для зберігання тегів.

4.3.2 Діаграма класів програмного забезпечення

Детальну діаграму класів програмного забезпечення подано у графічному матеріалі. Нижче наведена таблиця із коротким смисловим описом класів.

Таблиця 4.1 – Опис класів програмного продукту

Компонент	*Клас*	*Опис*
front-end	Receiver	Перший клас із послідовності обробки звуку, відповідає за запис звуку із користувацького мікрофона та початкову фільтрацію.
front-end	Processor	Другий клас із послідовності обробки звуку, відповідає за задачу вторинної фільтрації звуку та ідентифікації й фільтрування відрізків корисного сигналу.
front-end	Analyser	Третій клас із послідовності обробки звуку, відповідає за задачу точного знаходження початків корисного сигналу та підрахунок цільових точок графіка та інтегральних характеристик.
front-end	MeasurementsClient	Клас HTTP клієнта для бекенд API вимірювань, відповідає за виконання CRUD запитів.
front-end	TagsClient	Клас HTTP клієнта для бекенд API тегів, відповідає за виконання CRUD запитів.
front-end	GoogleOauthClient	Клас HTTP клієнта для Google Oauth API, відповідає за верифікацію токенів доступу користувачів на стороні клієнта.

back-end	MeasurementsController	Клас HTTP контролера для API вимірювань, відповідає за обробку та валідацію вхідних HTTP запитів та відсилання HTTP відповідей.
back-end	TagsController	Клас HTTP контролера для API тегів, відповідає за обробку та валідацію вхідних HTTP запитів та відсилання HTTP відповідей.
back-end	MeasurementService	Клас логіки сутності вимірювань, відповідає за створення об’єктів сутності та виконання CRUD операцій над ними, включаючи специфічну бізнес логіку, та роботу із колекцією в базі даних.
back-end	TagService	Клас логіки сутності тегів, відповідає за створення об’єктів сутності та виконання CRUD операцій над ними, включаючи специфічну бізнес логіку, та роботу із колекцією в базі даних.
back-end	FirestoreDataSource	Клас, що забезпечую підключення до бази даних та надання клієнта для виконання запитів.

4.3.3 Специфікація функціональних модулів

Повну сигнатуру методів класів програмного забезпечення можна побачити на діаграмі класів у графічному матеріалі. Проте, в силу використання мови Javascript та Typescript для створення продукту, які є мульти-парадигменими, деяка логіка була розроблена не у формі класів, а у формі функціональних модулів, які можна побачити на діаграмі компонентів у графічному матеріалі. Нижче наведена специфікація функцій, що містяться в цих модулях.

Таблиця 4.2 – Специфікація функціональних модулів

Компонент	*Модуль*	*Сигнатура функції*	*Опис*
front-end	utils	getFPS(): number	Дозволяє отримати частоту оновлення кадрів дисплею користувача.
front-end	utils	normalise(data: number[], max: number): number[]	Нормалізує переданий масив чисел від 0 до 1 за переданим максимумом.
front-end	utils	soundGate(data: number[], gate: number): number[]	Обрізає всі значення хвилі, амплітуда яких менша за переданий ліміт, перетворюючи їх на 0.
front-end	utils	getAudioStats(data: number[]): AudioWithMeta	Повертає інтегральні характеристики переданого масиву чисел, такі як середнє арифметичне, максимум, мінімум, кількість нулів та інші.
front-end	utils	splitAudioChunks(buffer: number[], sampleRate: number, chunkDuraion: number): number[][]	Поділяє передані дані аудіо-запису на відрізки заданого часу, використовуючи sampleRate для переведення часу в семпли.
front-end	utils	writeAudioToChannel(buffer: AudioBuffer, data: number[], channelIndex = 0) : void	Записує переданий масив даних до заданого каналу переданого аудіо-буферу.
front-end	utils	absoluteToRelative(list: number[],	Переводить масив абсолютних координат до відносних

		blockSize: number): number[][]	координат у відрізках заданої довжини.
front-end	graphics	drawWaveForm(data: number[], ctx: CanvasContext, height: number, width: number, fillColor: string, strokeColor: string, barWidth: number): void	Відмальовує хвильову форму звукового сигналу за допомогою Javascript Canvas API.
front-end	graphics	drawHistogram(data: number[], ctx: CanvasContext, height: number, width: number, fillColor: string, strokeColor: string, barWidth: number): void	Відмальовує гістограму за переданими даними за допомогою Javascript Canvas API.
back-end	utils	authHook(req: FastifyRequest, rep: FastifyReply): void	Функція, що реєструється як Fastify хук і верифікує аутентифікацію HTTP запитів.

4.3.4 Діаграма послідовності

Діаграми послідовностей наведено у графічному матеріалі роботи. Було виділено дві основних послідовності використання продукту, які покривають всі

варіанти використання: послідовність вимірювання та послідовність перегляду записів вимірювань.

4.4 Інфраструктура програмного забезпечення

В розділі 4.1.3 було описано технічні засоби, що використовувались для створення інфраструктури, такі як Cloud Run, Firestore та Google Cloud Storage. Беручи до уваги ці засоби та інфраструктурну діаграму можна сказати, що інфраструктура продукту повністю розроблена на основі Google Cloud Platform, що також називається Cloud Native.

Для розгортання фронтенд та бекенд додатків були використані окремі сервіси Cloud Run, база даних також розгорнута в окремому середовищі, що реалізує мікросервісну архітектуру з інфраструктурної точки зору.

4.4.1 Автоматизація розгорнення інфраструктури

Окрім цього для автоматизації розгортання інфраструктури продукту було використано Terraform IaS із віддаленим збереженням стану в Cloud Storage у поєднанні із підходом GitOps. GitOps являється підходом до створення інфраструктури, ключовим принципом якого є збереження стану інфраструктури у GitHub репозиторії.[33]

Автоматизація складається із чотирьох основних компонентів, як видно на рисунку 4.7:

- GitHub репозиторій, що зберігає IaS код
- Cloud Build Trigger та Cloud Build пайплайн
- Cloud Storage Bucket, що зберігає стан Terraform

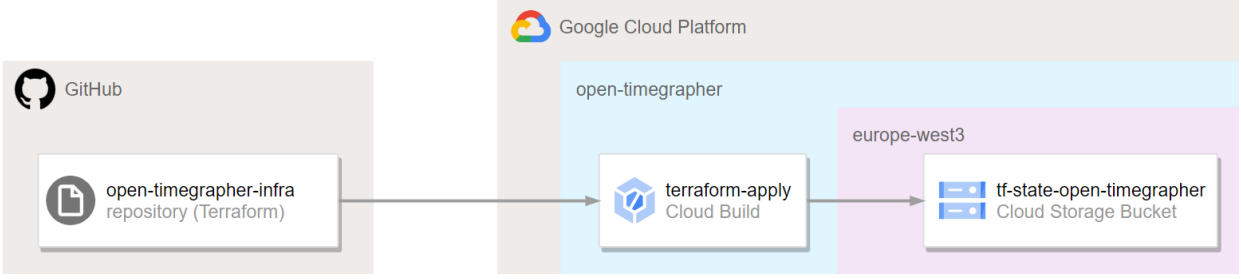

Рисунок 4.7 – Автоматизація розгортання інфраструктури

Адміністратор інфраструктури продукту може написати IaS Terraform код, що описує нові інфраструктурні ресурси, або змінити код конфігурації існуючих інфраструктурних ресурсів, після чого завантажити нову версію коду до репозиторія GitHub, де цей код може розглядатись та перевірятись власником продукту. Після підтвердження валідності коду, адміністратор створює та завантажує до репозиторія новий тег спеціального формату, що містить версію інфраструктури і виглядає як apply-v0.0.0. До репозиторія GitHub підключена інтеграція із Cloud Build і як тільки новий тег завантажується Cloud Build Trigger, що знаходиться всередині GCP, вловлює цю подію (на яку він налаштований завчасно) та запускає Cloud Build пайплайн, який всередині запускає контейнер образу Terraform, ініціалізує Terraform, який налаштований на збереження стану в Cloud Storage Bucket звідки він витягує поточний стан, далі пайплайн виконує команди terraform для внесення змін до інфраструктури, при чому Terraform автоматично знаходить різницю між поточним станом та бажаним станом у коді та після успішного виконання усіх змін в інфраструктурі, зберігає нову версію стану до Cloud Storage.

Цей підхід є максимально зручним та простим, хоч його налаштування і є складною задачею, він зберігає велику кількість часу у перспективі та допомагає уникати людських помилок. З точки зору користувача створення інфраструктури перетворюється на просте написання коду та його завантаження до репозиторія GitHub.

4.4.2 Автоматизація розгорнення продукту

Подібним чином до розділу 4.4.1 також було автоматизовано розгортання додатків самого продукту, що можна побачити на рисунку 4.8. Відмінність полягає в тому, що у цьому випадку Cloud Build Trigger налаштовані на подію завантаження коду до головної гілки GitHub репозиторія master, та запускають Cloud Build пайплайн, який будує образ додатку за допомогою Docker та зберігає цей образ в Artifact Registry, після чого дає команду на розгортання нової ревізії сервісу Cloud Run із новою версією образу додатка.[34]

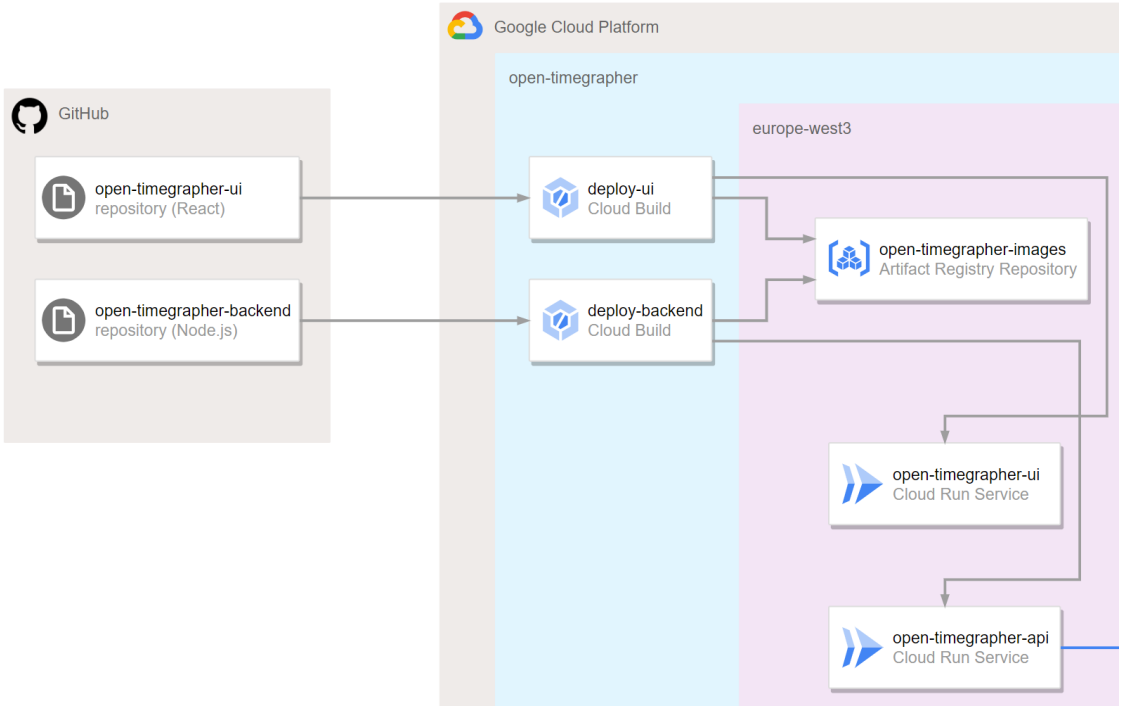

Рисунок 4.8 – Автоматизація розгортання додатків продукту

4.4.3 Інфраструктурна діаграма

Детальну діаграму інфраструктури Google Cloud Platform подано у графічному матеріалі роботи.

Висновок до розділу

В цьому розділі було детально розглянуто усі технічні засоби використані для створення продукту, у підсумку для написання коду додатків було використано Javascript/Typescript стек, React в ролі основного фронтенд фреймворка та Fastify в ролі основного бекенд фреймворка. Також було широко застосовано функціонал Web Audio API та Media API Javascript для роботи зі звуком. Для розгортання продукту та його інфраструктури була використана хмарна платформа GCP та такі основні продукти як Cloud Run та Firestore. Для створення інфраструктури був використаний інструмент IaS Terraform із провайдером для GCP. Нарешті для реалізації CI/CD продукту було використано такі основні інструменти як GitHub та Cloud Build.

Також у розділі було обґрунтовано вибір мікросервісної архітектури продукту та описано SPA архітектуру фронтенд додатку і архітектурний стиль

REST для додатку бекенд. Були надані різні архітектурні діаграми які описують структури архітектурних компонентів, класів програмного забезпечення та послідовностей виконання дій у системі. Було описано класи програмного забезпечення та надано специфікацію функціональних модулів додатків.

В останній частині було описано інфраструктуру продукту та надано детально інфраструктурну діаграму на основі GCP. Додатково було описано застосовані способи автоматизації розгортання інфраструктури за допомогою IaS, Terraform, GitOps та CloudBuild, і розгортання додатків продукту в Cloud Run за допомогою Cloud Build, Docker та Artifact Registry.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для того щоб розпочати роботу із додатком достатньо відкрити в браузері публічний URL фронтенд мікросервісу, що автоматично генерується Cloud Run. Після завантаження додатку, користувач опиниться на головній сторінці, рисунок 5.1, звідки в нього є два варіанти використання додатку: перегляд існуючих записів таймграфера різних годинників, та початок вимірювання годинника.

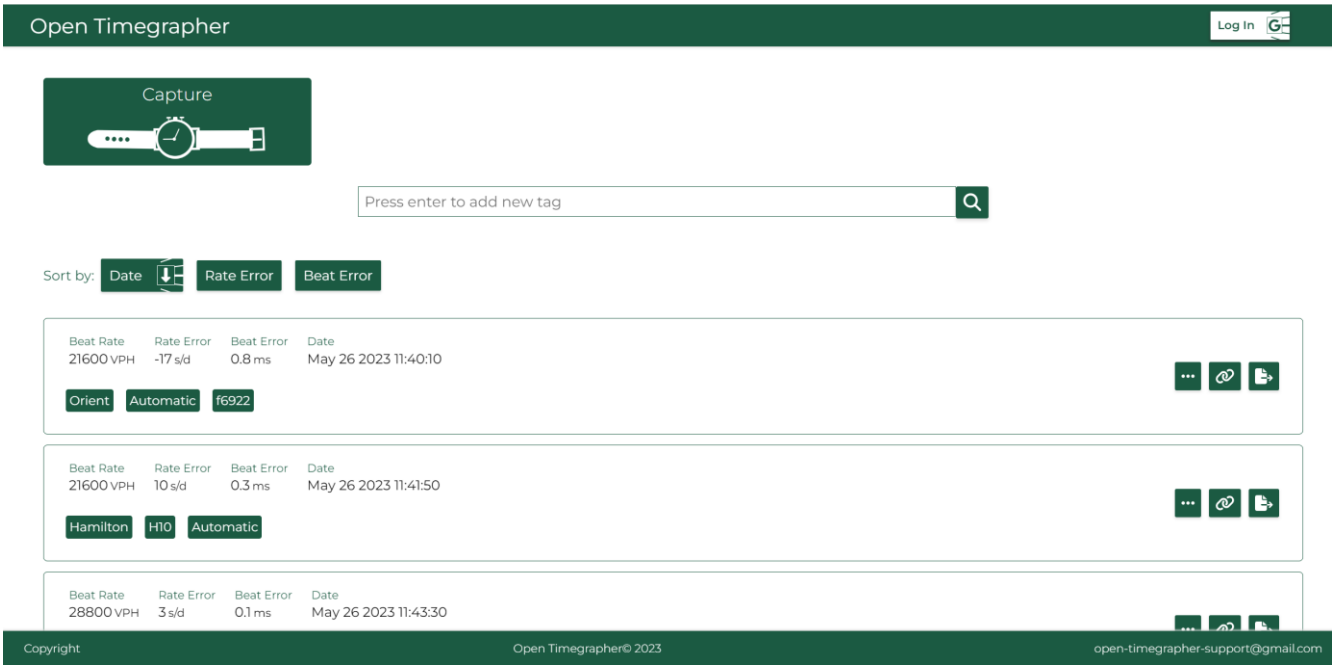

Рисунок 5.1 – Головна сторінка додатку

Для перегляду існуючих результатів вимірювання таймграфера, користувач маже використовувати елементи управління зображені на рисунку 5.2.

Рисунок 5.2 – Елементи управління переглядом вимірювань

1. Поле для вводу тегів, за якими проводиться пошук
2. Вибрані для пошуку теги, вони можуть видалятись за допомогою кнопки на кожному з тегів у вигляді хреста.

3. Кнопки сортування, користувач може сортувати записи вимірювань за датою створення, помилкою ходу та помилкою такту у висхідному та нисхідному порядках. Для вибору атрибута сортування користувач повинен натиснути на відповідну кнопку, повторні натискання цієї кнопки переключатимуть порядок сортування, який відображається іконкою стрілки на кнопці.

4. Після вибору тегів для пошуку та сортування, користувач повинен натиснути кнопку пошуку для завантаження вимірювань за критеріями пошуку.

5. Кнопка для відкриття сторінки деталей вимірювання, що містить графіки та характеристики.

6. Кнопка для копіювання посилання на вимірювання у буфер обміну.

7. Кнопка для копіювання посилання на об’єкт із даними вимірювання із бекенд REST API.

Для початку процесу вимірювання годинника, користувач повинен натиснути на велику кнопку «Capture»(рисунок 5.1). Після цього додаток відкриє сторінку звукозапису, рисунок 5.3.

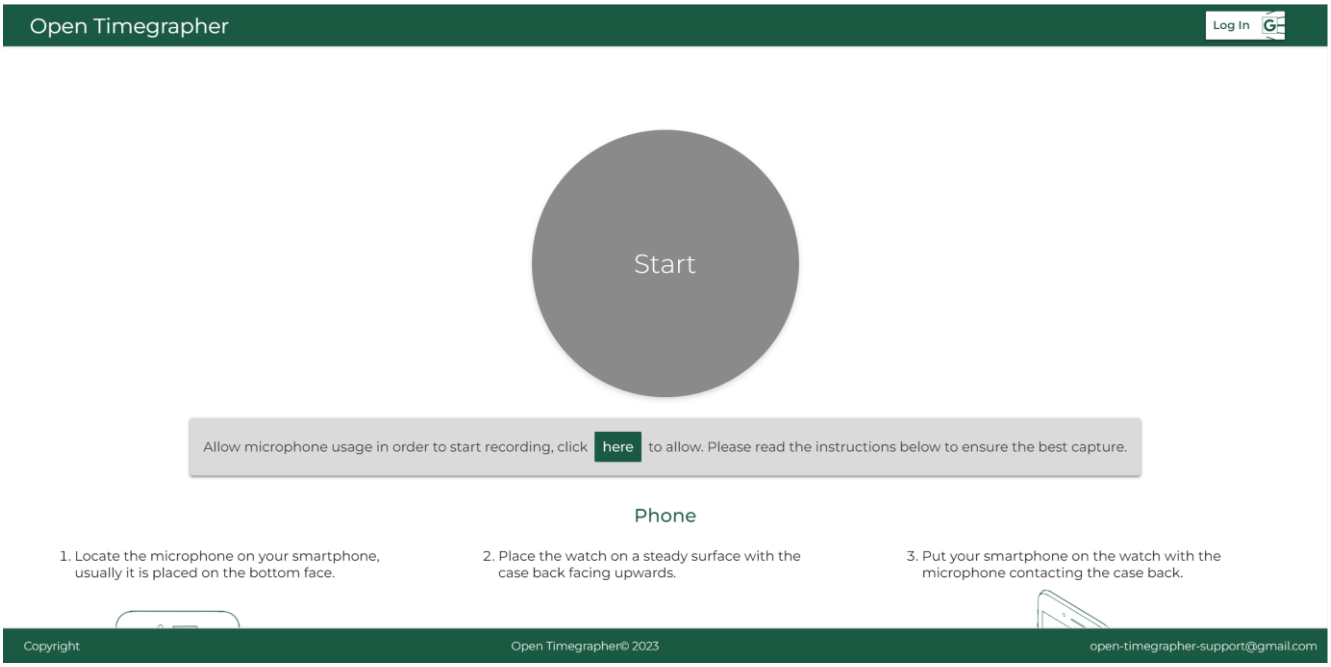

Рисунок 5.3 – Сторінка звукозапису

Далі користувач повинен ознайомитись із інструкціями для правильного запису звуку роботи годинника, що представлені нижче на тій самій сторінці. Інструкції представлені для використання мікрофону як телефону, так і ноутбука,

рисунок 5.4. Дані інструкції є критично важливими для коректної роботи додатку, оскільки звук роботи годинника є дуже тихим.

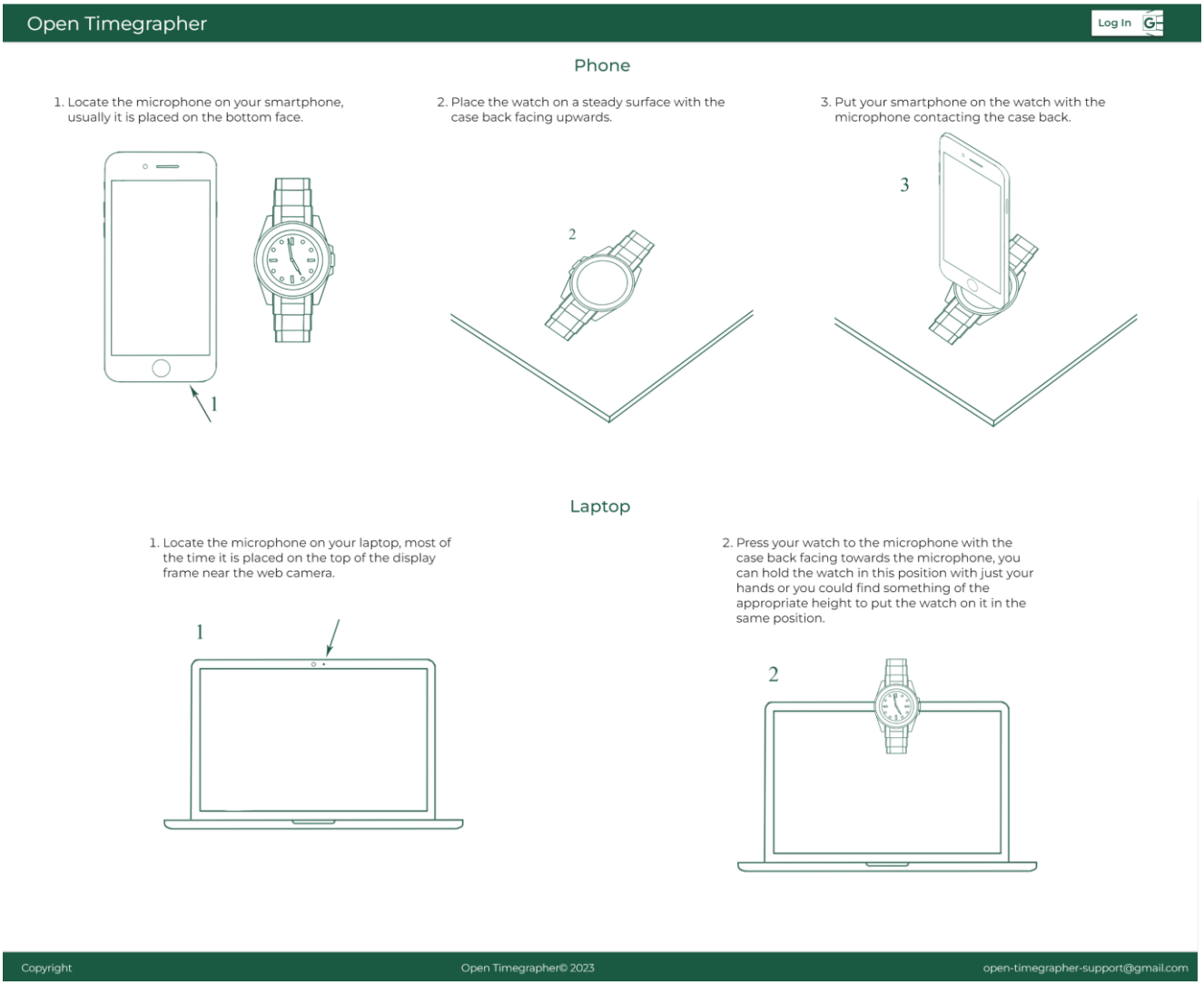

Рисунок 5.4 – Інструкції для звукозапису

Далі користувач повинен натиснути кнопку для дозволу використання мікрофону і погодитись із вспливаючим вікном, рисунок 5.5.

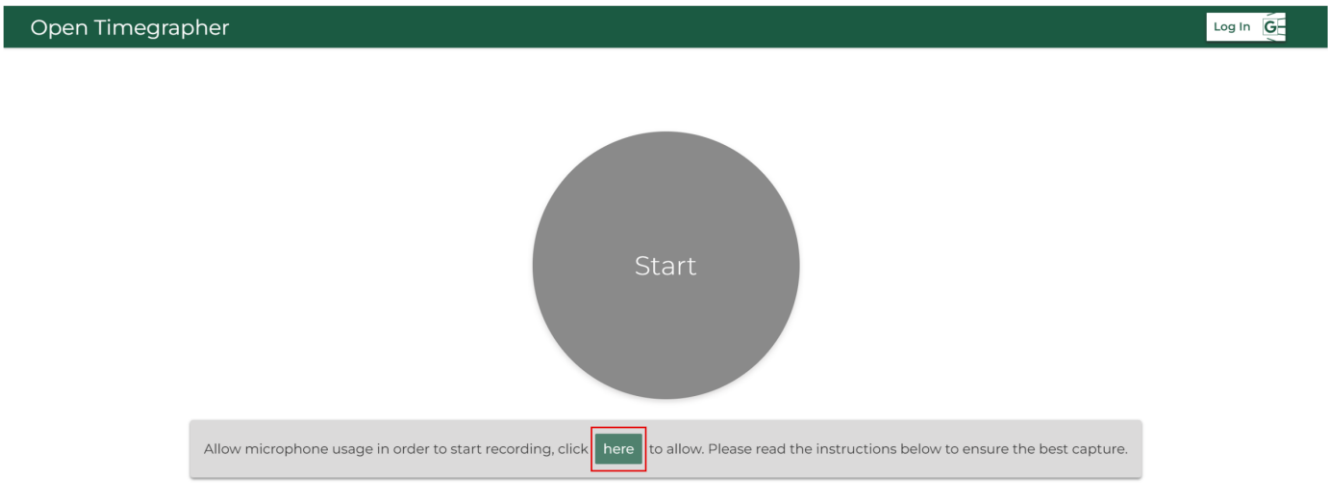

Рисунок 5.5. Дозвіл на використання мікрофона

Після цього сторінка на короткий час матиме доступ до вбудованого мікрофона, а також буде розблоковано кнопку «Start» для початку запису. Далі користувач повинен підготувати годинник для вимірювання та по готовності натиснути кнопку «Start», рисунок 5.6.

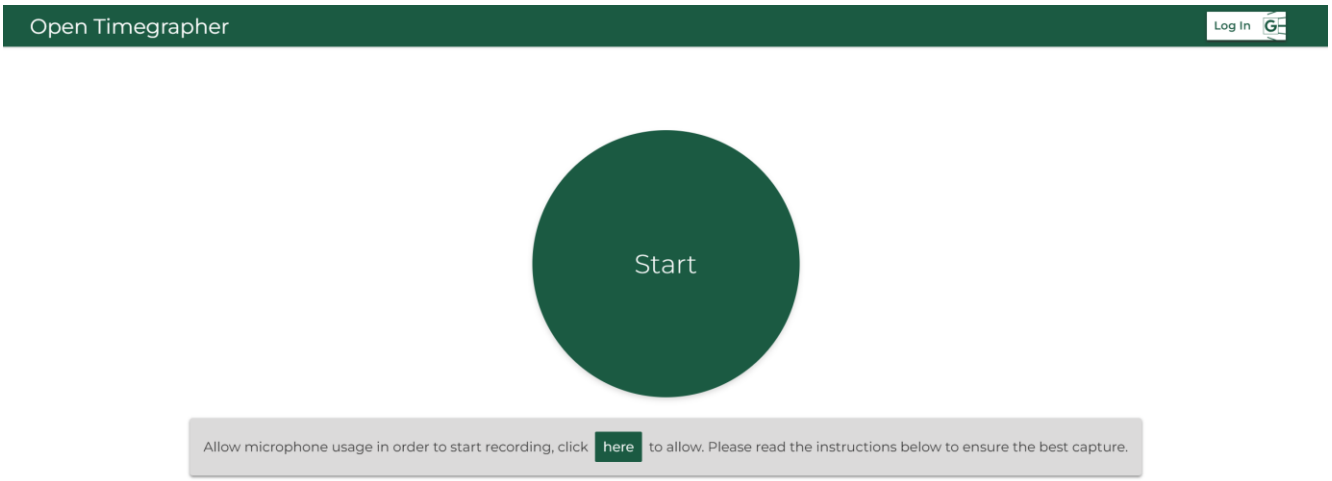

Рисунок 5.6 – Готовність до початку запису

Додаток почне запис фіксованої довжини в 40 секунд, рисунок 5.7, під час цього необхідно максимально обмежити будь-які навколишні шуми, а краще забезпечити умови повної тиші. При цьому найменший рух годинника під час вимірювання призведе до дуже сильного шуму у звукозаписі, через те що цей рух призведе до тертя прямо біля мікрофону, тому рекомендується, слідуючи інструкціям забезпечити стабільне зафіксоване положення годинника перед початком запису. У випадку випадкового шуму необов’язково дочікуватись кінця запису, можна натиснути на кнопку «Retry» для того, щоб почати процес заново.

Рисунок 5.7 – Процес запису звуку роботи годинника

Після завершення таймеру, додаток автоматично зупинить звукозапис та відкриє сторінку перевірки звукозапису, рисунок 5.8, також буде автоматично закрито доступ до потоку даних із мікрофона.

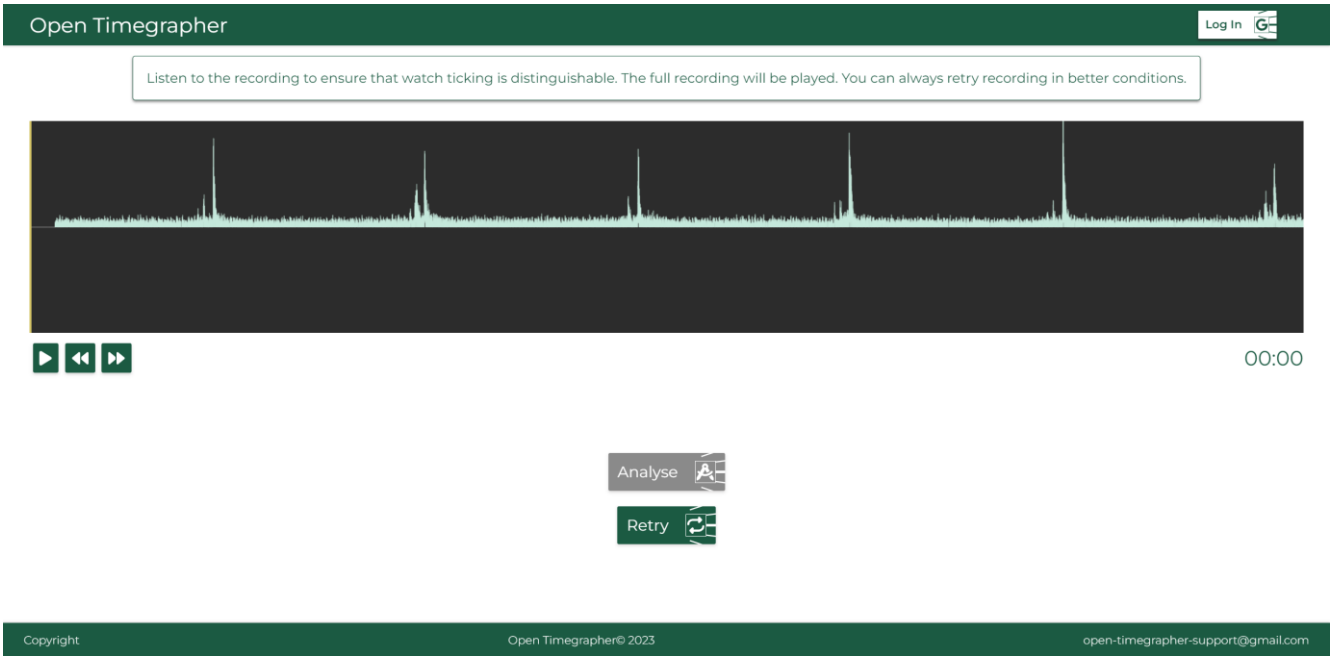

Рисунок 5.8 – Сторінка перевірки запису звуку роботи механізму

Наступним кроком користувач повинен натиснути кнопку «Play» для повного прослуховування зробленого запису, у випадку відчутних сторонніх шумів на слух або візуально на відображенні хвильової форми плеєра рекомендується зробити запис знову, хоча аналізатор сигналу має функціонал передбачений для

усунення нечисленних сторонніх звуків. У випадку якісного запису усі «тіки» повинні бути чітко видними на хвильовій формі, паралельно із програванням запису до користувача, додаток також буде обробляти звуковий сигнал і в успішному випадку відобразить знайдену частоту ходу(beat rate) механізму, рисунок 5.9, це є дуже хорошим показником, та майже повністю забезпечує успішність кінцевого результату.

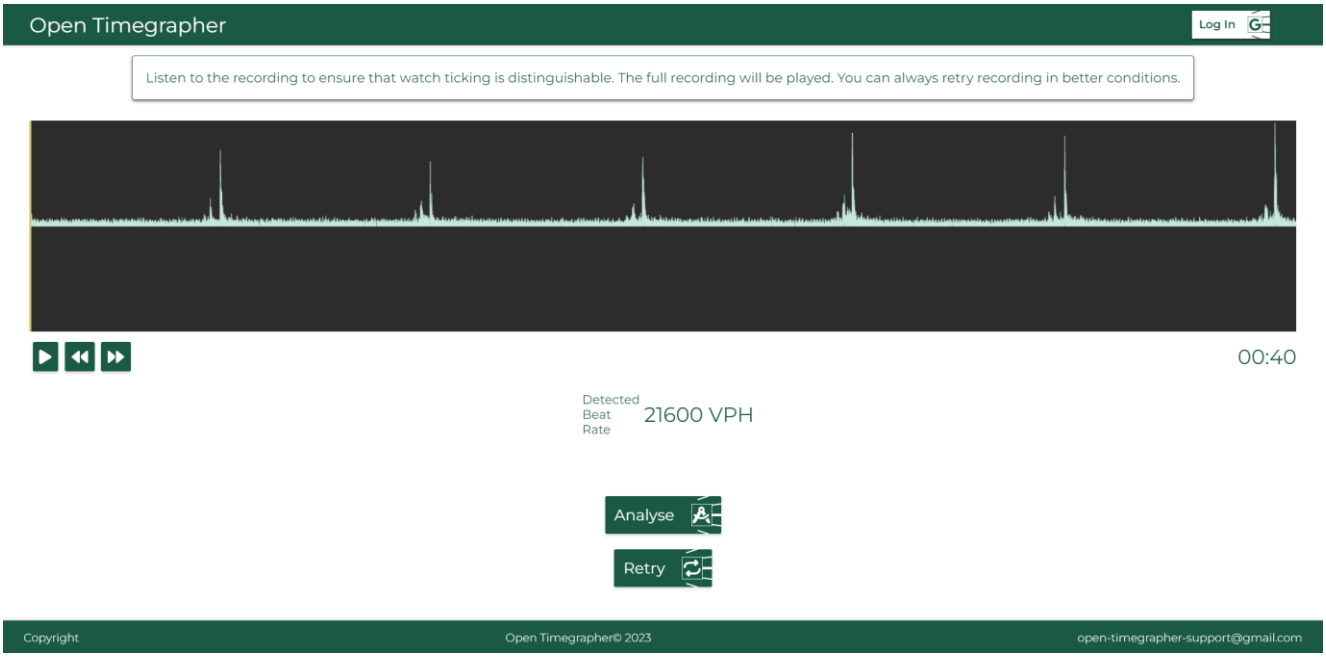

Рисунок 5.9 – Успішне розпізнавання частоти ходу та всіх «тіків»

Проте користувач повинен бути уважним, додаток може видавати хибні результати, оскільки підрахунок частоти ходу є досить складним, тому навіть якщо було відображено частоту ходу, але при цьому візуально та на слух присутня велика кількість шумів, потрібно переробити запис.

У неуспішному ж випадку додаток відобразить частоту ходу як 0 та не розблокує кнопку аналізу, рисунок 5.10. Можливі різні варіанти помилок, такі як неможливість вловлювання ходу(зображена на рисунку), ідентифікація присутності завеликої кількості сторонніх шумів, та неможливість визначення початку ходу у записі. В будь-якому з цих випадків, користувач повинен зробити запис знову у кращих умовах, натиснувши кнопку «Retry».

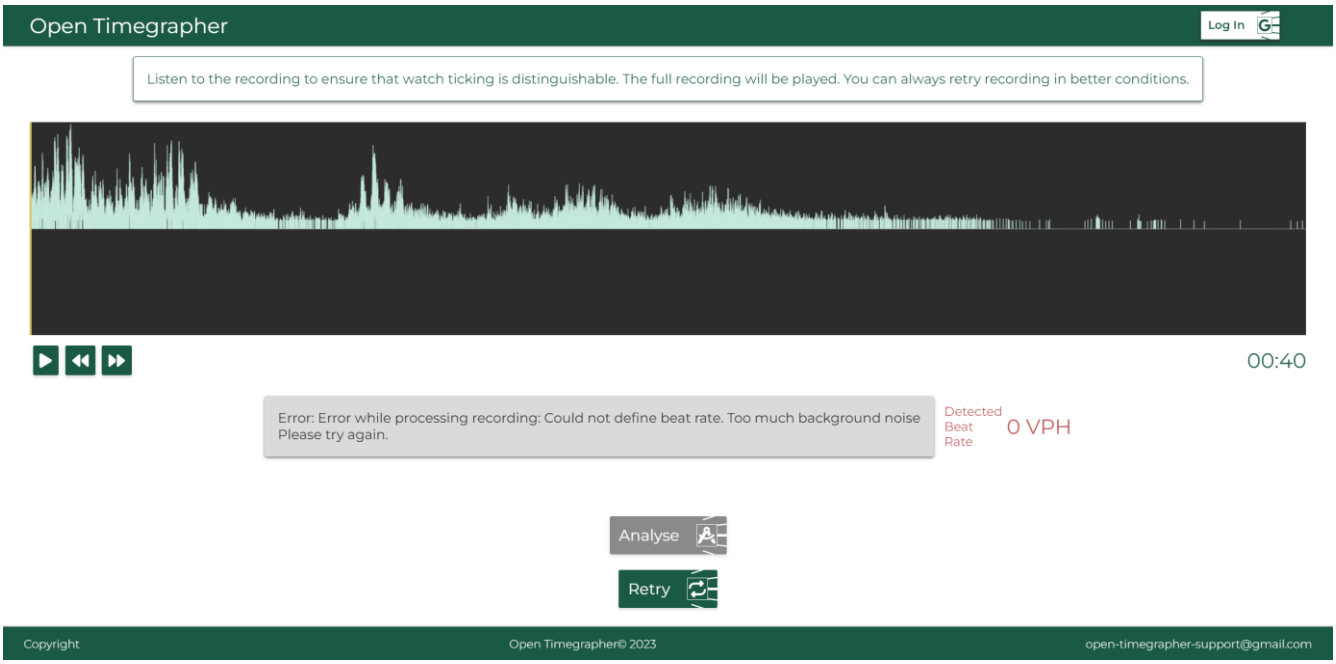

Рисунок 5.10 – Неможливість вловлювання ходу механізму

В разі успішного запису, наступним кроком користувач повинен натиснути на кнопку «Analyse», яка відкриє останню сторінку цього варіанту використання, рисунок 5.11, де буде відображено створений графік таймграфера, форму синтезованого звукового сигналу «тіка», що може бути корисною для проведення аналізу та обраховані характеристики механізма.

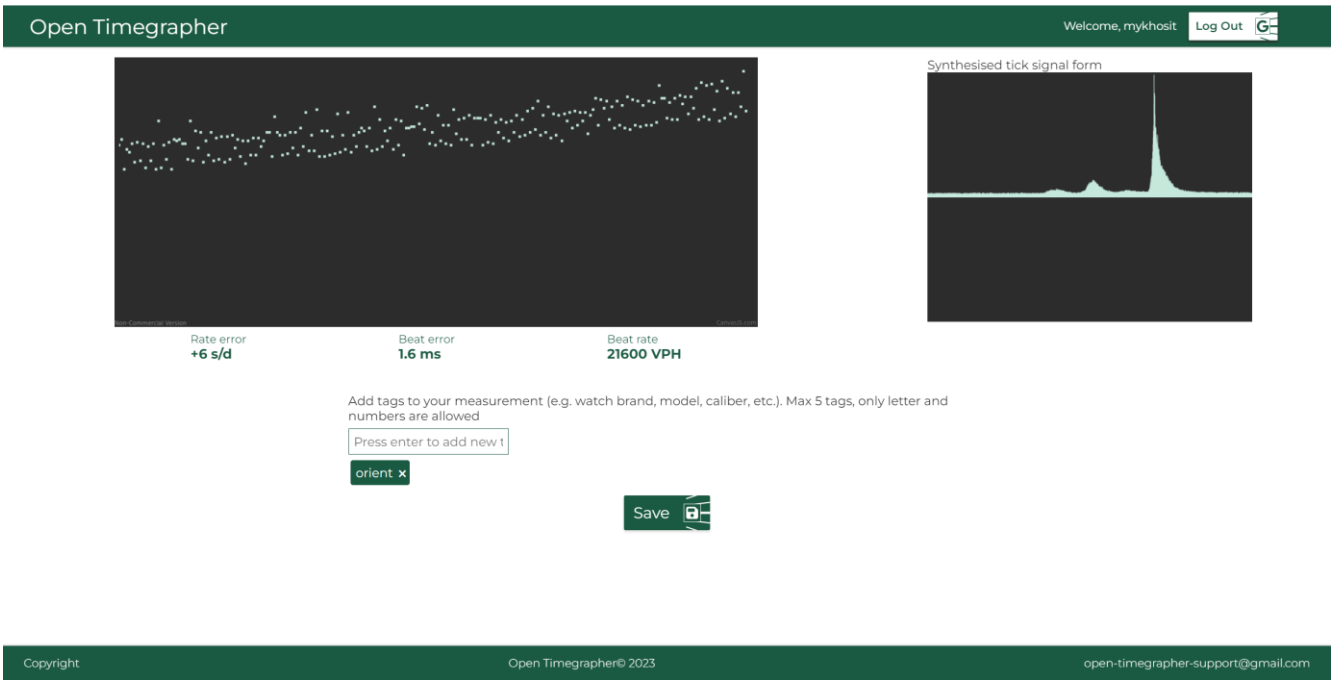

Рисунок 5.11 – Результуючі графіки та можливість збереження запису

Додатково, у разі випадкового або намірного відкриття неіснуючої сторінки, додаток відобразить сторінку із повідомленням про відсутність такої сторінки та можливістю повернутись на головну, рисунок 5.12.

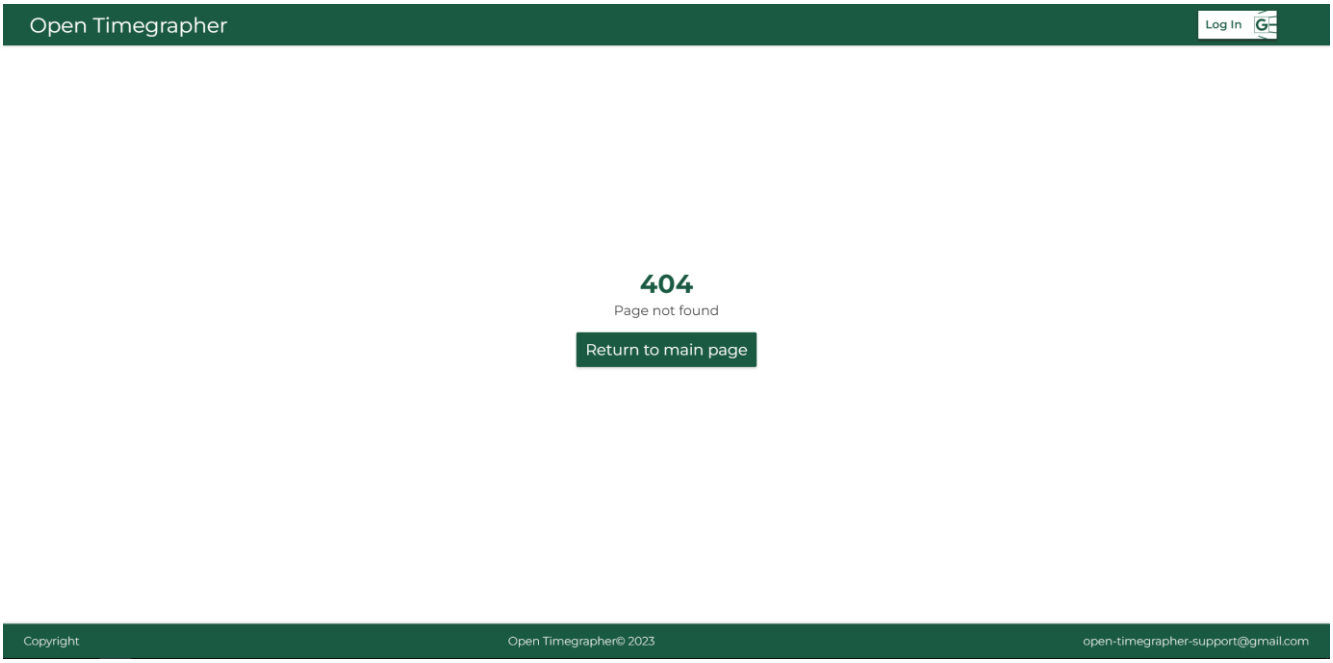

Рисунок 5.12 – Неіснуюча сторінка

5.2 Тестування програмного продукту

Після розробки програмного продукту було проведено тестування за допомогою ручного проходження тест-кейсів, створених на основі функціональних вимог до продукту, усі тест-кейси було успішно провалідовано, нижче наведений список перевірених тест-кейсів для кожного сценарію використання продукту.

Таблиця 5.1 – Тест-кейс до функціональної вимоги 1

Номер вимоги	1
Опис	Аутентифікація користувача
Передумови	Неаутентифікований користувач
Кроки виконання	1. Натиснути кнопку «Log In» 2. Вибрати обліковий запис Gamil у вікні аутентифікації Google 3. Ввести пароль до облікового запису

Очікуваний результат	Додаток відображає ім’я користувача, що відповідає його обліковому запису Gmail та не повертає ніяких помилок.
Фактичний результат валідний	так

Таблиця 5.2 – Тест-кейс до функціональної вимоги 3

Номер вимоги	3
Опис	Автоматичне визначення частоти ходу механізму
Передумови	Пройдено етап запису звуку роботи механізму
Кроки виконання	1. Натиснути кнопку «Play» 2. Очікувати 40 секунд до кінця програвання звукозапису
Очікуваний результат	Додаток відображає визначену частоту ходу механізму в окремому полі
Фактичний результат валідний	так

Таблиця 5.3 – Тест-кейс до функціональної вимоги 4

Номер вимоги	4
Опис	Запис звукового сигналу роботи годинника
Передумови	Відкрито сторінку запису звуку
Кроки виконання	1. Натиснути кнопку для надання дозволу на використання мікрофона 2. Натиснути кнопку «Start» 3. Очікувати 40 секунд
Очікуваний результат	Додаток автоматично починає та завершує запис звуку та відкриває сторінку із плеєром для прослухування звукозапису

Фактичний результат валідний	так

Таблиця 5.4 – Тест-кейс до функціональної вимоги 5

Номер вимоги	5
Опис	Зберігання форматованого запису
Передумови	Відкрито головну сторінку та за допомогою елементів контролю пошуку знайдено потрібний запис.
Кроки виконання	1. Натиснути кнопку для отримання посилання на запис в API 2. Вставити скопійоване посилання на API у браузер та перейти за ним
Очікуваний результат	Відкривається HTTP відповідь від бекенд API, що містить стандартний об’єкт запису
Фактичний результат валідний	так

Таблиця 5.5 – Тест-кейс до функціональної вимоги 6

Номер вимоги	6
Опис	Обрахування характеристик механізму
Передумови	Було прослухано та верифіковано звукозапис на сторінці верифікації, частота ходу механізму визначена правильно.
Кроки виконання	1. Натиснути кнопку «Analyse»
Очікуваний результат	Додаток відкриває сторінку аналізу та відображає в окремих полях характеристики помилки такту та помилки ходу.

Фактичний результат валідний	так

Таблиця 5.6 – Тест-кейс до функціональної вимоги 7

Номер вимоги	7
Опис	Обрахування характеристик механізму
Передумови	Було виконано запис звуку механізму та користувач знаходиться на сторінці перевірки запису
Кроки виконання	1. Натиснути кнопку «Analyse»
Очікуваний результат	Перед кроком 1 плеєр на сторінці перевірки запису відображає хвильову форму звуку. Після кроку 1 відкривається сторінка аналізу із графіком таймграфера
Фактичний результат валідний	так

Таблиця 5.7 – Тест-кейс до функціональної вимоги 8

Номер вимоги	8
Опис	Прослуховування запису звуку роботи механізму
Передумови	Було виконано запис звуку механізму та користувач знаходиться на сторінці перевірки запису
Кроки виконання	1. Натиснути кнопку «Play» плеєра хвильової форми 2. Очікувати 40 секунд до завершення програвання запису
Очікуваний результат	Під час програвання запису з вихідного звукового пристрою можна чути «тікання» механізму та після завершення програвання користувачу доступні кнопки для перемотки запису хвильової форми назад та вперед.

Фактичний результат валідний	так

Таблиця 5.8 – Тест-кейс до функціональної вимоги 9

Номер вимоги	9
Опис	Введення мета тегів
Передумови	Аутентифікований користувач знаходиться на сторінці аналізу і сторінка не відображає ніяких помилок обробки даних.
Кроки виконання	1. Ввести бажаний мета тег для запису в текстове поле 2. Натиснути кнопку «Save» для збереження запису
Очікуваний результат	Під час введення тегів у текстове поле, система автоматично пропонує найбільш близькі за буквами існуючі теги. Запис зберігається до бази даних разом із обраними тегами.
Фактичний результат валідний	так

Таблиця 5.9 – Тест-кейс до функціональної вимоги 11

Номер вимоги	10
Опис	Перегляд записів
Передумови	Користувач знаходиться на головній сторінці
Кроки виконання	1. Ввести в текстове поле пошуку тег, за яким буде проводитись пошук. 2. Натиснути кнопку пошуку
Очікуваний результат	Додаток відображає список записів, що відповідають критеріям пошуку.

Фактичний результат валідний	так

Таблиця 5.10 – Тест-кейс до функціональної вимоги 12

Номер вимоги	12
Опис	Отримання деталей запису
Передумови	Користувач знаходиться на головній сторінці
Кроки виконання	1. Натиснути на кнопку детального перегляду запису
Очікуваний результат	Додаток відкриває сторінку із всіма записами, при цьому URL сторінки побудований таким чином, щоб можна було вручну змінити ідентифікатор, що відкриє інший запис
Фактичний результат валідний	так

Висновок до розділу

В цьому розділі було розібрано технологічні аспекти роботи продукту. Було описано детальне керівництво користувача, приділяючи особливу увагу інструкціям щодо проведення якісного запису звуку роботи годинникового механізму, оскільки цей етап є ключовим в роботі додатка.

Також було створено набір тест-кейсів для перевірки відповідності системи до функціональних вимог, та успішно перевірено їх за допомогою ручного проходження.

ВИСНОВКИ

Головною ціллю даної роботи було створення додатку із функціоналом таймграфера у формі безкоштовного веб-додатку, з метою зробити таймграфер доступним для ширшої аудиторії користувачів, оскільки усі існуючі на сьогоднішній день аналоги є мобільними додатками, що зав’язані на одній операційній системі та здебільшого є платними. Актуальність розробки була описана у роботі і пов’язана зі збільшенням популярності механічних годинників як у первинному, так і у вторинному продажі та прогнозах на значне збільшення ринку механічних годинників у найближчі роки. Водночас було взято за ціль створити додаток, який за своєю точністю вимірів не буде сильно поступатись у точності мобільним конкурентам.

В цілому в результаті розробки програмного продукту даної роботи було досягнуто поставлених цілей, та реалізовано досить складний функціонал таймграфера, у веб-додатку, який може працювати у великій кількості браузерів, як мобільних, так і десктопних версіях, головним інструментом, який сприяв досягненню цілей були стандартні браузерні API Javascript, такі як Web Audio API та Media API для роботи зі звуком. Проте, точність вимірювань фінального продукту, хоч і є досить хорошою, все ж таки поступається прямим конкурентам, здебільшого це пов’язано із обмеженістю даної роботи у часі та дуже великою складністю роботи із реальним фізичним сигналом. За умови більш глибокого дослідження звукового сигналу із фізичної точки зору, особливо з точки зору амплітудно-частотної характеристики, можна значно покращити як основний алгоритм роботи інструмента, так і його точність, проте, як було згадано вище, ця робота була обмежена часом, а також своєю специфікою – настільки глибокі фізичні дослідження відносяться до сфери скоріше прикладної фізики, аніж розробки інформаційних систем. Тим не менш, було проведено всеохоплююче дослідження сигналу на основі фізичної моделі та розроблено алгоритм, який використовує як фізичний апарат, такий як частотні фільтри, так і математичний апарат, такий як теорія ймовірностей та випадкові величини, завдяки цьому поєднанню вдалося розробити працюючий алгоритм за досить короткий час.

З точки зору розробки інформаційних систем, у процесі роботи було розроблено повноцінну інформаційну платформу, яка охоплює усі основні навички сфери IT та повний життєвий цикл програмного продукту. По-перше, було проведено аналіз ринку та предметної області, після чого чітко сформовано цілі розробки та створено функціональні вимоги для майбутнього продукту. По-друге, було проведено дослідження фізичної моделі із подальшою розробкою та постановкою математичної задачі і створення алгоритму для вирішення цієї задачі. По-третє, було проведено дослідження та аналіз технологічних засобів розробки та вибрано найкращі із доступних та найсучасніших із них для програмної реалізації основного алгоритму, зокрема було використано AudioWorklet із Web Audio API, специфікацію якого було вперше презентовано тільки у 2018 році і з цієї причини на сьогоднішній день майже не існує доступної документації для його використання, тому цей проект також робить внесок в суспільний досвід використання новітніх API мови Javascript, крім цього було обрано такі засоби як Typescript, Node.js, Fastify, React та інші. По-четверте, було продумано та створено архітектуру програмного продукту із використанням таких сучасних архітектурних принципів як Cloud Native, мікросервісна архітектура, SAP та REST, а також продумано та створено дизайн графічного інтерфейсу користувача. По-п’яте, було створено дизайн інфраструктури для реалізації архітектури продукту, беручи за основу сучасні принципи хмарних ресурсів та Google Cloud Platform, як основної інфраструктурної платформи. По-шосте, було створено повний програмний код для реалізації алгоритму та архітектури, включаючи розробку бекенд додатку та фронтенд додатку. По-сьоме було автоматизовано процеси розгортання інфраструктури за допомогою сучасних принципів IaS, GitOps та інструменту Terraform, а також створено CI/CD пайплайн для автоматичного розгортання додатків продукту із використанням таких засобів як Cloud Build та Artifact Registry. Нарешті, останнім кроком було створено тест-кейси для перевірки результуючого продукту на відповідність функціональним вимогам та пройдено їх.

Підсумовуючи, можна сказати, що даний дипломний проект поєднує в собі достатньо глибокі наукові дослідження із демонстрацією на практиці повного

циклу розробки сучасної інформаційної системи із використанням сучасних принципів та інструментів. Можна сказати, що було повністю досягнуто загальних цілей розробки даного дипломного проекту та програмного продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Watches Industry Statistics & Analysis
   URL: https://www.davosa-usa.com/blogs/story-time/luxury-watches-industry-statistics-industry-analysis (дата звернення: 01.05.2023)
2. Mechanical Watch Market - Trends Forecast Till 2028
   URL: https://www.delvens.com/report/mechanical-watch-market-trends-forecast-till-2028 (дата звернення: 02.05.2023)
3. Susanne Samuelsson. How Often Does Your Watch Need to be Serviced?
   URL: https://www.thewatchpages.com/how-often-does-your-watch-need-to-be-serviced/ (дата звернення: 02.05.2023)
4. David Flett. Collector Guide. Interpreting Timegrapher Results
   URL: https://www.beyondthedial.com/post/collector-guide-interpreting-timegrapher-results/ (дата звернення: 03.05.2023)
5. Wikipedia. Data-flow Diagram
   URL: https://en.wikipedia.org/wiki/Data-flow_diagram (дата звернення: 10.05.2023)
6. Watchfinder&Co. Getting Technical: Swiss Lever Escapement
   URL: https://www.watchfinder.co.uk/articles/getting-technical-swiss-lever-escapement (дата звернення: 15.05.2023)
7. Witschi Electronic Ltd. Measuring Technology and Troubleshooting for Watches. 2010
   URL: https://www.javiergutierrezchamorro.com/wp-content/uploads/2022/06/amplitud_lift_angle_beat_error_witschi_training_course.pdf (дата звернення: 16.05.2023)
8. STEPHEN COFFIN. Spatial frequency analysis of block letters does not predict experimental confusions. 1978
   URL: https://link.springer.com/content/pdf/10.3758/BF03214297.pdf (дата звернення: 16.05.2023)

9. Thomas Most. Approximation of complex nonlinear functions by means of neural networks. 2005
URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ac871aa11b390f72d561349ae0114539ac2bccb5 (дата звернення: 17.05.2023)
10. Частотні характеристики
URL: https://studfile.net/preview/5685680/page:5/ (дата звернення 17.05.2023)
11. Є. А. Коробов. МОДЕЛЮВАННЯ ФІЗИЧНИХ ПРОЦЕСІВ В ЕЛЕКТРОННИХ LR-ФІЛЬТРАХ. 2019
URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/76628/1/Korobov_%20harmonic_signal.pdf;jsessionid=E39F59A223E83B62A04F29C60703ED97 (дата звернення: 17.05.2023)
12. Jhon Gibson. Reason: Dynamics Effects. 2013
URL: https://cecm.indiana.edu/361/rsn-dynamics.html (дата звернення: 17.05.2023)
13. ТЕОРІЯ ЙМОВІРНОСТЕЙ. Барабаш О.В., Мусієнко А.П., Свинчук О.В. 2021
URL: https://ela.kpi.ua/bitstream/123456789/42046/1/Navch_Posib_Teor_Ymovirn_BarabashO_MusienkoA_SvynchukO.pdf (дата звернення: 08.05.2023)
14. Розподіл радіохвиль за діапазонами.
URL: https://studfile.net/preview/3757704/page:4/ (дата звернення: 21.05.2023)
15. Mozilla. Web Audio API
URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (дата звернення: 10.05.2023)
16. About Node.js
URL: https://nodejs.org/en/about (дата звернення: 20.05.2023)
17. Hooks
URL: https://www.fastify.io/docs/latest/Reference/Hooks/ (дата звернення: 20.05.2023)

18. Validation and Serialization
URL: https://www.fastify.io/docs/latest/Reference/Validation-and-Serialization/ (дата звернення: 20.05.2023)
19. Plugins
URL: https://www.fastify.io/docs/latest/Reference/Plugins/ (дата звернення: 20.05.2023)
20. Pino
URL: https://www.npmjs.com/package/pino?activeTab=readme (дата звернення: 20.05.2023)
21. React
URL: https://uk.legacy.reactjs.org/ (дата звернення: 21.05.2023)
22. Web Audio API
URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (дата звернення: 21.05.2023)
23. Video and Audio APIs
URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Video_and_audio_APIs (дата звернення: 21.05.2023)
24. Canvas API
URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (дата звернення: 21.05.2023)
25. What is Cloud Run
URL: https://cloud.google.com/run/docs/overview/what-is-cloud-run (дата звернення: 26.05.2023)
26. Cloud Firestore
URL: https://firebase.google.com/docs/firestore (дата звернення: 26.05.2023)
27. Using OAuth 2.0 to Access Google APIs
URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 26.05.2023)

28. What is Terraform?
URL: https://developer.hashicorp.com/terraform/intro (дата звернення: 26.05.2023)
29. Artifact Registry overview
URL: https://cloud.google.com/artifact-registry/docs/overview (дата звернення: 26.05.2023)
30. Overview of Cloud Build
URL: https://cloud.google.com/build/docs/overview (дата звернення: 26.05.2023)
31. Microservice architecture style
URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (дата звернення: 01.06.2023)
32. Representational state transfer
URL: https://en.wikipedia.org/wiki/Representational_state_transfer (дата звернення: 01.06.2023)
33. Managing infrastructure as code with Terraform, Cloud Build, and GitOps
URL: https://cloud.google.com/docs/terraform/resource-management/managing-infrastructure-as-code (дата звернення: 02.06.2023)
34. Deploying to Cloud Run using Cloud Build
URL: https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run (дата звернення: 02.06.2023)