Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка програмної системи селективного аналізу та синтезу звукових сигналів

Виконав: студент II курсу
групи 1ПІ-18 м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Стахов Л. П.
(прізвище та ініціали)

Керівник: к. т. н., доц. Рейда О. М.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Освітньо-кваліфікаційний рівень – магістр

Спеціальність 121 – інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

Романюк О. Н.

«___» _____ 2019 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стахову Леоніду Петровичу

1. Тема роботи – Розробка програмної системи селективного аналізу та синтезу звукових сигналів.

Керівник роботи: Рейда Олександр Миколайович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: частота дискретизації аудіо-сигналу; вбудована аудіо-карта з частотою дискретизації 192 kHz і разрядністю ЦАП 24 біта; браузер Google Chrome з підтримкою Web Audio API, двухядерний процесор Intel з підтримкою технології Hyper-threading.

Вихідні методи для модифікації: Web Audio API, бібліотека Rxjs, фреймворк Angular.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз сучасного стану питання та постановка задачі; розробка методу реалізації селективного аналізу та синтезу звукових сигналів; розробка програмного забезпечення;

тестування програмного додатку; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: актуальність теми; мета, об’єкт та предмет дослідження; основні задачі; результати порівняльного аналізу програм-аналогів; методи обробки звукових сигналів; засоби розробки програми; програмний код додатку; інтерфейс програмного продукту; результати тестування; оцінювання комерційного потенціалу розробки; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Рейда О. М., к.т.н., доцент кафедри ПЗ		
5	Бальзан М. В., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасного стану питання та постановка задачі	04.09.2019 – 29.09.2019	Виконано
2	Розробка методу реалізації селективного аналізу та синтезу звукових сигналів	30.09.2019 – 26.10.2019	Виконано
3	Розробка програмного забезпечення	27.10.2019 – 3.11.2019	Виконано
4	Тестування програмного додатку	4.11.2019 – 12.11.2019	Виконано
5	Економічна частина	13.11.2019 – 17.11.2019	Виконано

Студент _____ Стахов Л. П.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Рейда О. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській кваліфікаційній роботі «Розробка програмної системи селективного аналізу та синтезу звукових сигналів» розроблено метод обробки звукових сигналів, який підвищив ефективність цифрового аналізу та синтезу звукових сигналів шляхом використання реактивного багатопотокового програмування у веб-додатках.

У ході виконання роботи було проаналізовано існуючі методи синтезу звукових сигналів та існуючі програмні аналоги. Було розроблено та перевірено роботу програмного додатку для обробки та синтезу звукових сигналів за допомогою таких засобів: мова програмування TypeScript, Web Audio API, Rxjs та Angular.

ABSTRACT

In master's qualification thesis on «Development software system for selective analysis and synthesis of audio signals. There were implemented different methods of processing audio signals, which increased the efficiency of digital analysis of audio signals by using multithreaded reactive programming approach in web applications.

During the work were analyzed different existing methods of synthesis of sound signals and existing software analogues.

The software application for processing and synthesizing audio signals was developed and tested using the following tools: TypeScript programming language, Web Audio API, Rxjs and Angular.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Цифрова обробка сигналів охоплює як технічні, так і програмні засоби. Основним її завданням є виділення сигналу на тлі шумів і перешкод різної фізичної природи, автоматичне розпізнавання, тобто класифікація та ідентифікація сигналу. Основними методами попередньої обробки сигналів є процес швидких дискретних перетворень, що реалізуються в різних функціональних базисах, методах лінійної алгебри, лінійної і нелінійної фільтрації [1].

Сфера використання цифрового звуку на даний час значно розширюється. Останнім часом із впровадженням WEB 2.0. відбувається інтеграція мультимедіа, і цифрового звуку зокрема в глобальну інформаційну мережу. Веб-програмування в наш час – це одна з найбільш перспективних сфер діяльності, а використання хмарних сервісів і онлайн додатків стає все більш популярним і необхідним. В результаті такої популярності збільшується кількість сучасних веб-стандартів, а технології веб-програмування дають можливість створювати все більш складні веб-додатки.

Тому, останнім часом можна спостерігати бурхливий ріст медіа-напрямків, пов’язаних із соціальними мережами: рух подкастерів, promodj, онлайн редакторів. Тому виникає цілком обґрунтована потреба в технологіях обробки цифрового звуку онлайн, які були б доступними широкому загалу аудиторії, не залежно від рівня технічної освіти та спеціальних знань.

Актуальність роботи. Актуальність визначена зростаючою тенденцією до широкого використання звукових сигналів в інтернет просторі, а отже, потребою створення методу і програмної системи, які дозволять підвищити швидкодію синтезу, аналізу та обробки звукових сигналів у веб-додатках.

Саме тому темою магістерської кваліфікаційної роботи було обрано розробку програмної системи селективного аналізу та синтезу звукових сигналів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та задачі дослідження. Метою магістерської кваліфікаційної роботи є підвищення ефективності цифрового аналізу та синтезу звукових сигналів шляхом використання реактивного багатопотокового програмування у веб-додатках.

Основними задачами дослідження є:

- дослідити тенденції розвитку технологій у сфері обробки звукових сигналів;
- проаналізувати сучасні тенденції у створенні веб програм для обробки звукових сигналів онлайн ;
- проаналізувати існуючі засоби для селективного аналізу і синтезу звукових сигналів;
- запропонувати покращення швидкодії селективного аналізу і синтезу звукових сигналів;
- розробити метод селективного аналізу і синтезу звукових сигналів;
- розробити програмний додаток;
- провести тестування додатку.

Об'єкт дослідження – процес селективного аналізу та синтезу звукових сигналів у цифровому вигляді.

Предмет дослідження – методи та засоби аналізу та синтезу звукових сигналів.

Методи дослідження. У процесі досліджень використовувались: методи цифрової фільтрації, спектрально аналізу, синтезу звукових сигналів, методи інтернет-технологій для роботи зі звуком у браузері, методи та принципи реактивного програмування для розробки методів створення багатопотокових веб-додатків.

Наукова новизна отриманих результатів.

1. Вперше запропоновано багатопотоковий метод селективного аналізу звукових сигналів у веб-додатках, особливість якого полягає у використанні паралельних багатопроцесорних методів, що дозволило підвищити швидкодію та оптимізувати використання технічних засобів.

2. Вперше запропоновано комбінований метод виконання синтезу звукових сигналів із використанням технологій реактивного та багатопотокового програмування Rxjs, що дозволяє підвищити швидкодію та оптимізувати використання технічних засобів.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі результатів запропоновано метод реалізації програмних систем селективного аналізу та синтезу звукових сигналів, що побудовані за принципом реактивного програмування з використанням паралельних багатопроцесорних методів

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: порівняльний аналіз javascript (JS) фреймворків для розробки мережевого журналу [2];

Апробація результатів роботи над проектом. Результати роботи над проектом доповідалися на XLVII Науково-технічній конференції викладачів та студентів ВНТУ, 21 березня 2018 року, м. Вінниця, на секції програмного забезпечення XLVIII Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії 14 – 15 березня 2019 р.

Публікації. Результати роботи над проектом були опубліковані у збірці доповідей XLVII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії [2].

Структура та обсяг роботи. Магістерська кваліфікаційна роботи складається зі вступу, чотирьох розділів, висновків, списку літератури, що містить 21 найменувань, 3 додатків. Робота містить 35 ілюстрацій, 29 формул, 7 таблиць.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз стану проблеми

Цифрова обробка сигналів (ЦОС) (digital signal processing) – це область обчислювальної техніки, що на сьогодні постійно розвивається та охоплює як програмні, так і технічні засоби. Одними з найбільш споріднених галузей для цифрової обробки сигналів є теорія інформації, яка включає в себе теорію оптимального прийому сигналів, основним завданням якої є виділення сигналу на фоні шумів і завад різної фізичної природи, і теорію розпізнавання образів, основним завданням якої є автоматичне розпізнавання, тобто класифікація та ідентифікація сигналу [3].

Сучасним альтернативним рішенням традиційної аналогової обробки сигналів є цифрова обробка. Основними якісними перевагами цифрової обробки є програмованість та функціональна гнучкість, а також можливість реалізації алгоритмів обробки будь-якої складності з високою точністю, що значно менше залежить від впливу дестабілізуючих факторів. Основними напрямками цифрової обробки сигналів є звукова локація, радіолокація, акустика, біомедицина, сейсмологія, зв'язок, системи передачі даних, багатьох інших.

Значним поштовхом у розвитку цифрової обробки сигналів стало винайдення в 1965 році нових ефективних алгоритмів для обчислення перетворення Фур'є. Ці алгоритми стали відомі як "швидке перетворення Фур'є" (ШПФ, fast Fourier transform). Перспективність використання ШПФ були значними з багатьох причин. Більшість алгоритмів для обробки сигналів, що використовувались раніше, потребували часу на обчислення, що був на декілька порядків більше від реального часу. Зазвичай це було зумовлено тим, що важливою складовою цифрової обробки сигналів був спектральний аналіз, і на той час ефективні засоби для його виконання ще були невідомі. Завдяки алгоритму швидкого перетворення Фур'є вдалося зменшити на декілька

порядків час обчислення перетворення Фур'є. Це дозволило виконувати складні алгоритми цифрової обробки сигналів у реальному часі. Крім того, враховуючи можливості реалізації алгоритму ШПФ на спеціалізованому цифровому пристрої, велика кількість алгоритмів цифрової обробки сигналів, які були непрактичними раніше, змогли знайти втілення та реалізацію [3].

Методами цифрової обробки сигналів є алгоритми або математичні співвідношення, за допомогою яких виконуються обчислювальні операції над цифровими сигналами. До таких методів належать алгоритми спектрально-кореляційного аналізу, цифрової фільтрації, модуляції та демодуляції сигналів та адаптивної обробки.

Засобами реалізації ЦОС є жорстка логіка, програмовані логічні інтегральні схеми, мікропроцесори загального призначення, мікроконтролери, персональні комп'ютери (комп'ютерна обробка сигналів) та цифрові сигнальні процесори [3].

Представлення звуку у цифровому вигляді перш за все цінне тим, що має здатність довготривалого зберігання і також можливість тиражування без втрати якості, однак перетворення сигналу з аналогової форми в цифрову і в зворотному напрямі все ж таки неминуче призводить до часткової втрати його якісної складової. Зокрема при квантуванні сигналу по рівню через округлення амплітуди до найближчого дискретного значення виникають неприємні на слух спотворення – гранулярний шум. Цей шум, на відміну від широкосмугового шуму, що вноситься помилками квантування, є гармонійним спотворенням сигналу, що стає найпомітнішим у верхній частині спектру.

При відновленні звуку з цифрового формату в аналоговий сигнал виникає проблема згладжування ступінчастої форми сигналу і придушення гармонік, що вносяться частотою дискретизації. Через певні недоліки амплітудно частотних характеристик (АЧХ) фільтрів може відбуватися або надмірне ослаблення корисних високочастотних складових, або недостатнє придушення цих перешкод. Гармоніки частоти дискретизації при поганому придушені

спотворюють, особливо на зрізі високих частот, форму аналогового сигналу, що створює враження "шорсткого", "брудного" звуку.

Обробка цифрового звуку виконується за допомогою математичних операцій, які застосовуються до окремих проміжків сигналу, або до груп проміжків різної довжини. Такі математичні операції можуть імітувати роботу традиційних аналогових засобів обробки:

• мікшування двох або декількох сигналів,

• складання,

• посилення та послаблення сигналу,

• множення на константу,

• модуляція,

• множення на функції;

або використовувати альтернативні методи, зокрема:

• розкладання сигналу в спектр (ряд Фур'є),

• корекція окремих частотних складових, і подальша зворотна "збірка" сигналу із спектру.

Обробка цифрових сигналів поділяється на дві складові: лінійну (у реальному часі) і нелінійну - обробка заздалегідь записаного сигналу. Лінійна обробка є більш вимогливою до швидкодії обчислювальної системи (процесора), тому іноді через неможливість поєднання необхідної швидкодії і якості використовується спрощена обробка зі зниженою якістю вихідного сигналу. Нелінійна обробка зазвичай не обмежена жорстко у часі, тому для неї можуть використовуватись обчислювальні засоби будь-якої потужності, а час обробки, особливо за умови високої якості вихідного сигналу, може сягати декількох хвилин або навіть годин.

Виконання цифрової обробки аудіо сигналу може виконуватись на універсальному процесорі або на спеціалізованому DSP (digital signal processor). Різниця між ними полягає у тому, що перший є орієнтованим на широкий клас задач (наукові, економічні, логічні, ігрові і та інші задачі), і містить великий

набір команд загального призначення, в якому переважають звичайні математичні і логічні операції.

В свою чергу DSP спеціально орієнтований на обробку сигналів і містить набори специфічних операцій для цієї цілі – перемноження векторів, складання з обмеженням, обчислення математичного ряду та інші. Тому виконання навіть нескладної цифрової обробки звуку на універсальному процесорі вимагає значної швидкодії і не завжди можливе у режимі реальному часі, в той час як навіть простий DSP зазвичай здатні виконувати навіть відносно складну обробку у реальному часі, а потужні DSP здатні виконувати якісну спектральну обробку одразу декількох сигналів.

Через свою спеціалізацію DSP рідко застосовуються самостійно – зазвичай пристрій обробки має універсальний процесор середньої потужності для управління всім пристроєм, прийому/передачі інформації, взаємодії з користувачем, і один або декілька DSP – для обробки звукового сигналу.

Веб-програмування в наш час – це одна з найбільш перспективних сфер діяльності. У сучасному світі, якщо у будь-якої компанії, бізнесу, магазину немає власного сайту, або веб-додатку, він значно відстає від своїх конкурентів. Стрімкий розвиток інтернету за останні пару років призвів до збільшення кількості нових хмарних-сервісів. Починаючи від первісних характеристик і основних сервісів, доступних користувачам (пошта і веб), інтернет розвинувся у велику всесвітню павутину з великою кількістю різних сервісів, технологічних рішень і став тим місцем, де зустрічаються практично бізнес-організації всього світу.

Однією з переваг використання інтернету є хмарні обчислення. Хмарні сервіси стають все більш популярними і необхідними. В результаті такої популярності збільшується кількість сучасних веб-стандартів, а технології веб-програмування дають можливість створювати все більш складні веб-додатки.

Звичайне програмне забезпечення, встановлене на персональному комп’ютері, виконує всі операції, використовуючи обчислювальні потужності локальної машини. У свою чергу хмарне програмне забезпечення виконує

більшу частину обчислень на потужному сервері, і відсилає клієнту тільки результати обчислень. «Хмарні обчислення» є онлайн-альтернативою тим процесам, які зазвичай виконуються стаціонарно.

Перевагами «хмарних обчислень» є зниження вимог до обчислювальної потужності локальної обчислювальної машини, відсутність потреби встановлення програмного забезпечення на локальну машину та онлайн доступ до програмного забезпечення.

Недоліками «хмарних обчислень» є недостатня безпека інформації користувача в інтернеті.

1.2 Порівняльний аналіз аналогів

Насьогодні на ринку програмного забезпечення для обробки звукових цифрових сигналів існує доволі велика кількість різноманітних програмних рішень. Розглянемо найбільш характерні програми-редактори цифрового звуку, та визначимо їх переваги та недоліки.

Adobe Audition (Cool Edit Pro) - професійний редактор, призначений для обробки аудіо і відеопродукції. Програма пропонує широкі можливості редагування, мікшування і обробки звукових спецефектів. В продукті поєднуються гнучкість технологічного процесу з простотою у використанні. Крім того, Adobe Audition може використовуватись як студія звукозапису завдяки гнучким і простим у використанні інструментам.

Основною перевагою цього програмного продукту є його висока функціональність. Основним недоліком є доволі високі вимоги до знання користувачем принципів будови і обробки цифрового звуку. Таким чином, вхідний поріг використання програми обмежується рівнем професійних знань користувача.

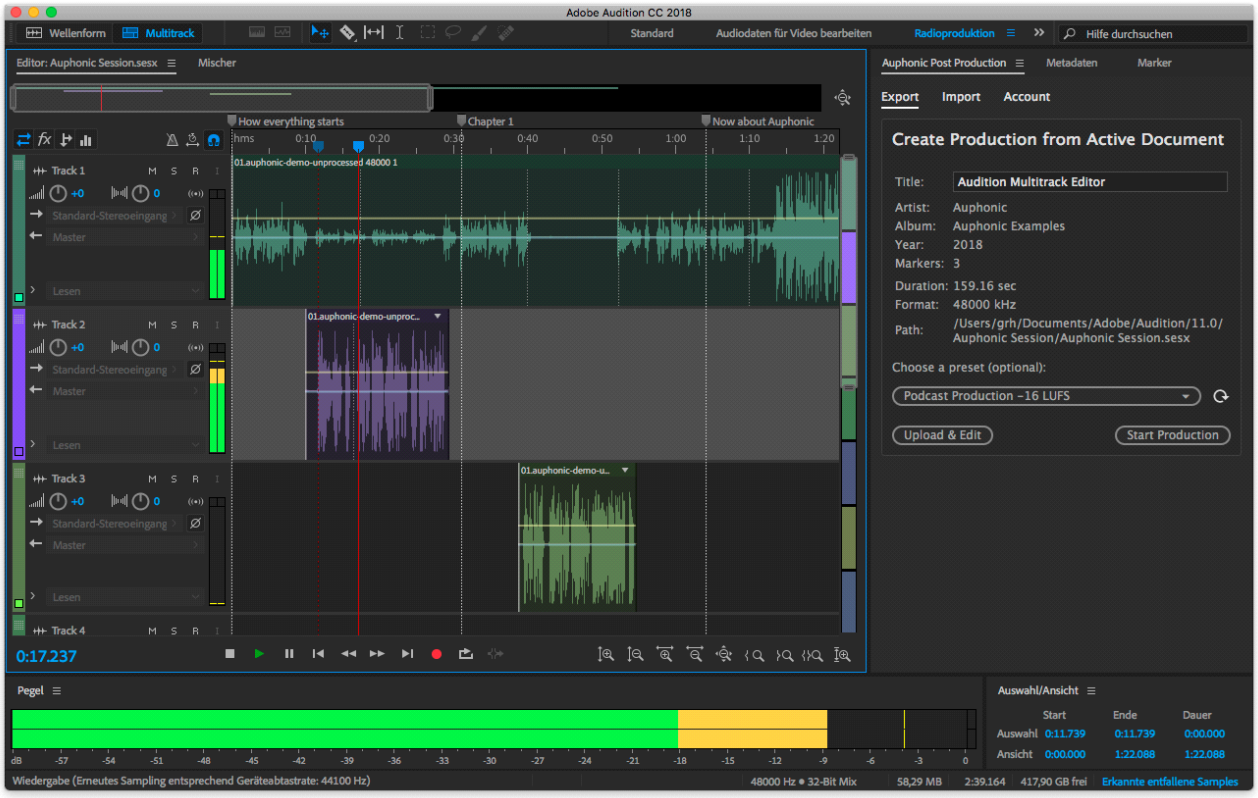


Рисунок 1.1. – Інтерфейс програми Adobe Audition

Audio Editor Pro - ще один аудіо-редактор, який дозволяє записувати і редагувати аудіо-файли форматів mp3, wma, wav із застосуванням великої кількості цифрових ефектів і фільтрів (доступні 20 ефектів і 6 фільтрів), а також є можливість конвертації файлу з одного формату в інший і копіювання аудіо-CD на жорсткий диск. Також підтримуються mp3 VBR кодек, Windows Media 9, ID3-теги і завантаження даних з CDDB.

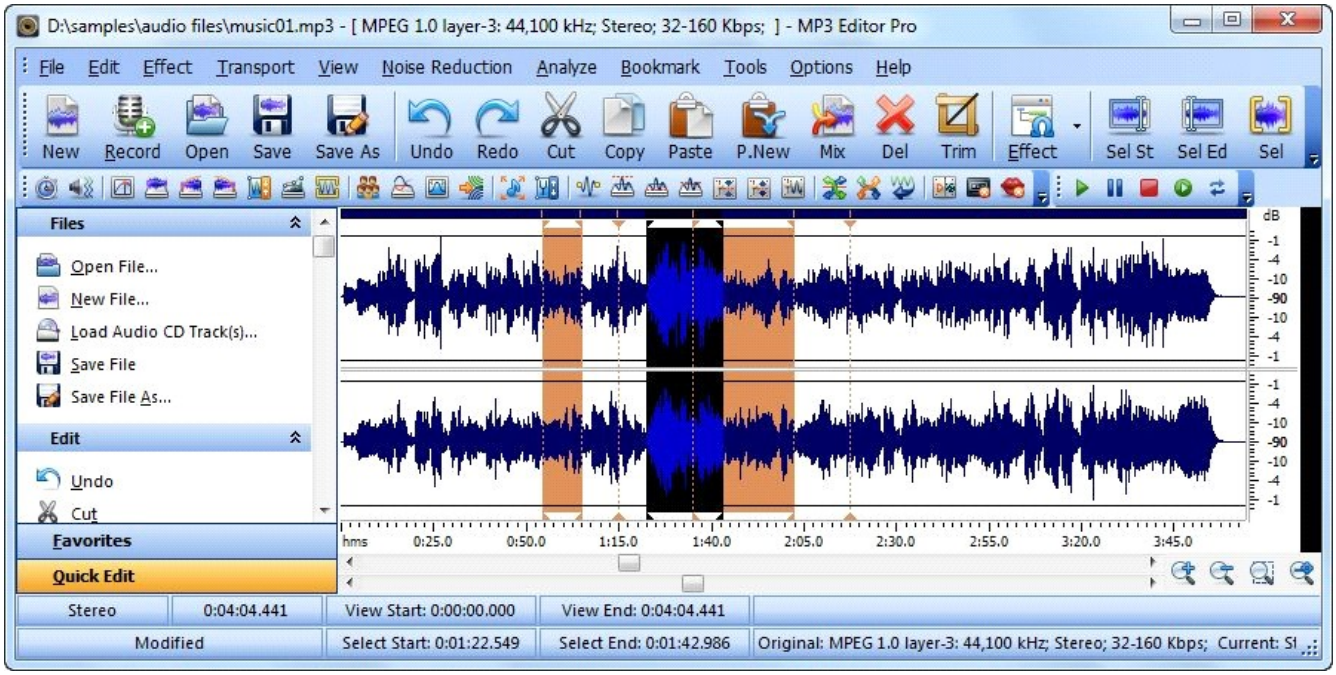


Рисунок 1.2. – Інтерфейс Audio Editor Pro

Дана програма є доволі простою у використанні, має дружній до користувача інтерфейс, однак суттєвим недоліком є необхідність придбання ліцензії, що робить програму не зовсім доступною для пересічного користувача.

При розробці програми необхідно скористатись досвідом розглянутих вище програмних засобів, і розробити з однієї сторони максимально спрощений і зрозумілий для кожного користувача програмний продукт. З іншої сторони, не зважаючи на простоту необхідно забезпечити виконання всього необхідного функціоналу при роботі із цифровим звуком. Тому слід одночасно враховувати обидва ці напрямки розробки.

В результаті розвитку інтернету і його можливостей на сьогоднішній день починають активно збільшуватись попит на онлайн-студії звукозапису. Це означає, що користувач має можливість звернутися до цього серверу задля реалізації запису, створення чи обробки звуку, не витрачаючи коштів на апаратуру та програмне забезпечення. Єдине, що потрібно – це підключення до Інтернету. Нинішній їх рівень дозволяє перенести функціональні можливості багатьох програм в браузер. Редагування аудіо та відео за допомогою

мережевих ресурсів тепер реальність, яка доступна будь-якому користувачеві мережі. Розглянемо найпопулярніші онлайн сервіси для онлайн обробки та мастерінгу звукового контенту:

1. IndabaMusic Mantis – це нова веб-база компанії DAW.

IndabaMusic Mantis – цифрова робоча станція для аудіо, яка виконана у вигляді Java-апплета. Можливості програми дозволяють записувати звук якості CD (44 кГц/16 біт), додавати аудіо-семпли з бібліотеки Indaba, яка містить у собі більш ніж 10000 якісних безкоштовних зразків, здійснювати за допомогою вбудованих ефектів обробку звуку, прописувати автоматизацію та зберігати отримані мікси на сервері Indaba. Також є можливість надання доступу для декількох учасників одночасно. Програма отримала елегантний дизайн і порівняно високу швидкість роботи, стабільність та зручність використання. За рівнем можливостей IndabaMusic наближається до таких професійних цифрових станцій, як Steinberg Cubase.

Рисунок 1.4. – Інтерфейс програми IndabaMusic Mantis

Основні особливості IndabaMusic Mantis:

- режим сесії;
- спільний запису, редагування з іншими користувачами в інтернеті;
- запис звука CD якості;
- бібліотека семплів.

2. AudioSauna – повнофункціональна аудіо станція для створення музики в Інтернеті.

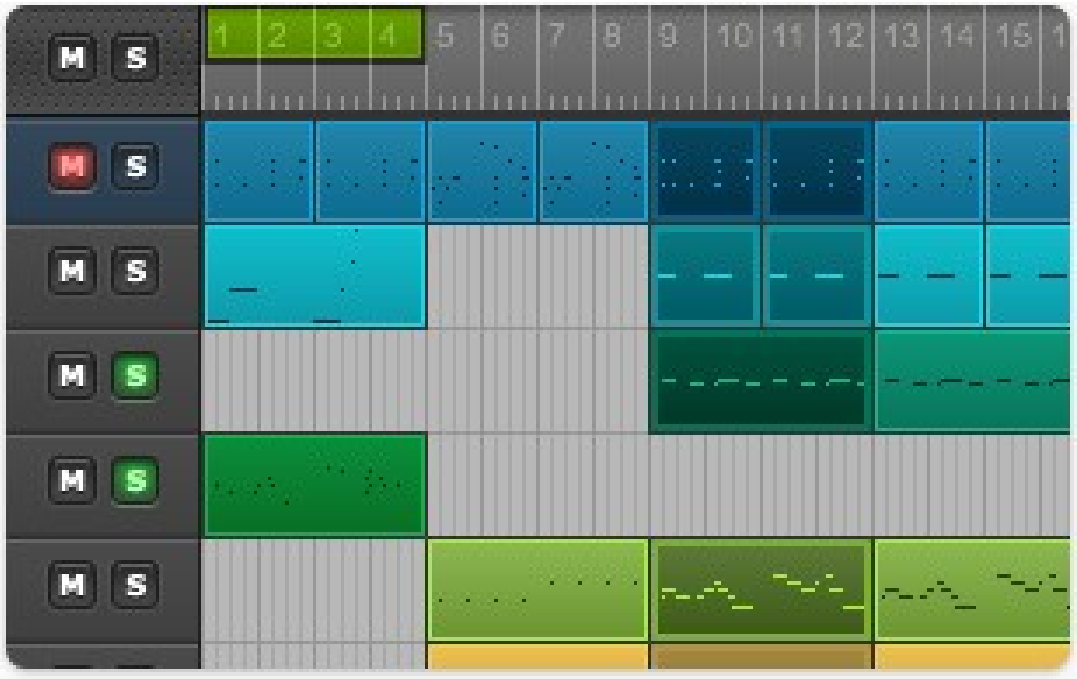


Рисунок 1.6 – Інтерфейс програми AudioSauna

AudioSauna – ще одна повнофункціональна аудіо-станція, яка має простий у використанні інтерфейс. Можливості програми дозволяють використовувати прямо у браузері декілька просунутих поліфонічних синтезаторів, семплерів і ефектів. Робота з доріжками проводиться на базі платформи JAVA, тому є доволі вимогливою до ресурсів ПК. Сайт пропонує користувачам досить широкий асортимент музичних інструментів на вибір, які допоможуть створити мелодію для майбутньої пісні.

Основні особливості:

- можлива робота як в браузері, так і встановлення на ПК;
- експорт композицій у форматі .WAV;

- не обов’язково створювати обліковий запис, для створення та збереження треків.

Порівняльні характеристики усіх аналогів наведено в таблиці 1.1.

Таблиця 1.1

Порівняння аналогів

Параметри/порівняльні характеристики	Запропоновані аналоги			
	Adobe Audition	Audio Editor Pro	IndabaMusic Mantis	AudioSauna
Можливість використовувати як веб додаток	- (0.5)	- (0.5)	+ (0.5)	+ (0.5)
Можливість синтезу сигналу	+ (0.6)	+ (0.6)	+ (0.6)	+ (0.6)
Наявність селективного аналізу	+ (0.4)	- (0.4)	+ (0.4)	- (0.4)
Використання потоків	+ (0.7)	- (0.7)	- (0.7)	- (0.7)
Не потребують професійних навичок	- (0.3)	+ (0.3)	+ (0.3)	+ (0.3)
Надання доступу до обробки, редагування декільком учасникам одночасно	- (0.4)	- (0.4)	+ (0.4)	- (0.4)
Доступ до всіх функціональних можливостей програми у безкоштовному режимі	- (0.4)	- (0.4)	+ (0.4)	+ (0.4)
Сума	1.7	0.9	2.6	1.8

Розглянуті вище аналоги, незважаючи на великий набір функцій, мають певні недоліки і не задовільнять усі вказані параметри (табл. 1.1).

На підставі вищевикладеного можна стверджувати, що новий метод і програмна система, що пропонується для розробки, вирішить задачу використання багатьох паралельних потоків для селективного аналізу і синтезу

звукових сигналів у веб додатках, і тому його розробка та впровадження є актуальною та доцільною.

1.3 Аналіз методів розв’язання поставленої задачі

Сигнал – це інформаційна функція, що несе в собі відомості про фізичні властивості, стан або поводження певної фізичної системи, об'єкта або середовища. Метою обробки сигналів можна вважати отримання певних інформаційних відомостей, що містяться у цих сигналах, а також перетворення цих відомостей у форму, зручну для сприйняття й подальшого використання. Поняття "аналіз" сигналів містить в собі не тільки математичні перетворення, але і означає процес отримання на основі цих перетворень висновків про специфічні особливості відповідних об'єктів або процесів [5].

Класифікація сигналів здійснюється на підставі істотних ознак відповідних математичних моделей сигналів. Сигнали поділяють на дві групи: детерміновані та випадкові (рис. 1.5).

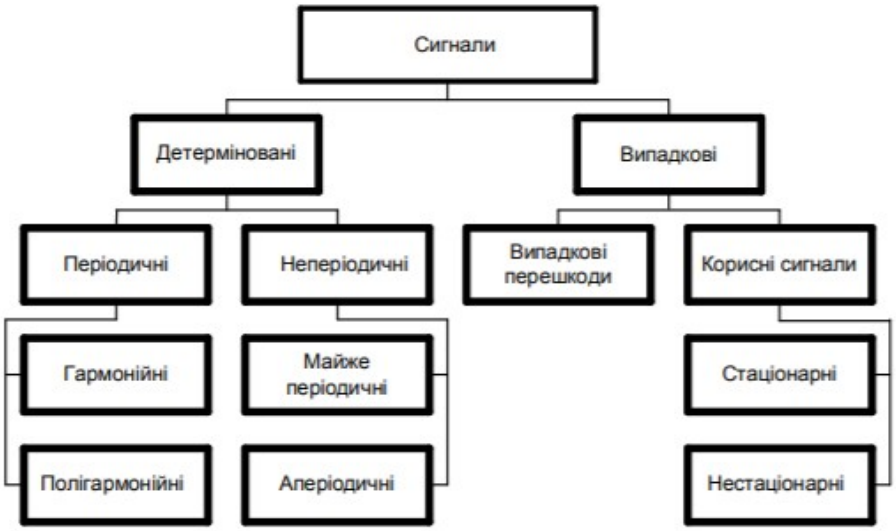


Рисунок 1.8 – Класифікація сигналів

Для перетворення безперервного аналогового сигналу в цифровий вигляд необхідно обрати раціональний крок дискретизації. Оптимальний крок

дискретизації встановлюється за допомогою теореми Котельникова, яка має важливе теоретичне та практичне значення. Відповідно до цієї теореми довільний сигнал *s(t),* спектр якого обмежений деякою частотою $F_в$, може бути повністю відновлений за послідовністю своїх відлікових значень, що слідують з інтервалом часу, причому інтервал дискретизації розраховується наступним чином [4]:

$$\Delta t = \frac{1}{2F_в}. \quad (1.1)$$

Інтервал дискретизації Δt та частоту *Fв* часто називають інтервалом і частотою Найквіста. Аналітично теорема Котельникова подається рядом

$$s(t) = \sum_{k=-\infty}^{\infty} s(k\Delta t) \frac{\sin\frac{\pi}{\Delta t}(t - k\Delta t)}{\frac{\pi}{\Delta t}(t - k\Delta t)}, \quad (1.2)$$

де k – номер відліку; $s(k\Delta t)$– значення сигналу в точках відліку.

Фізичний зміст цієї теореми стає зрозумілим, якщо розглянути спектри сигналів *S(t)* і $S_D(t)$.

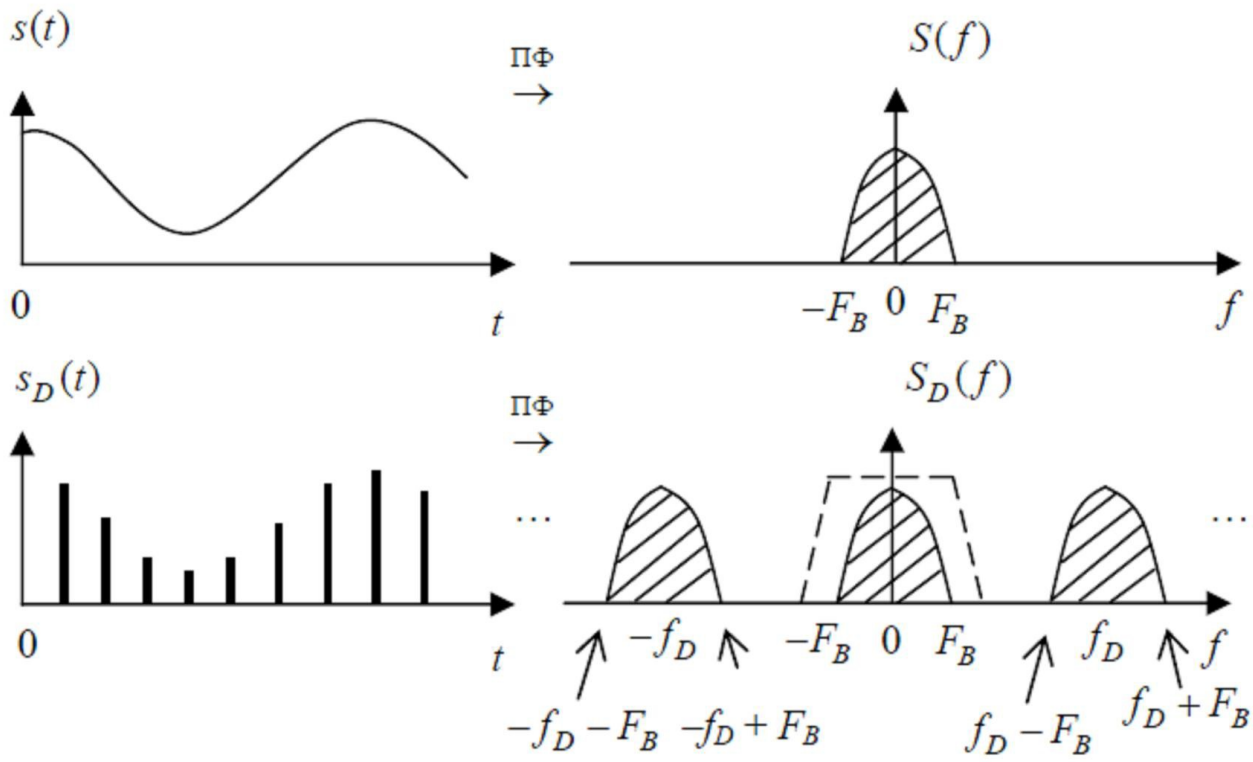


Рисунок 1.9 – Теорема Котельникова

На практиці ця теорема має важливе значення. Наприклад, відомо, що більшість звукових сигналів, які можуть бути сприйняті вухом людини, можна з деякою мірою точності обмежити спектром частот до 20 кГц. Це означає, що за умови дискретизації з частотою не менше 40 кГц вихідний аналоговий звуковий сигнал відновлюється достатньо точно. На практиці найчастіше використовується частота дискретизації 44100 Гц. Пристрій, який перетворює дискретний сигнал у аналогову форму, називається цифро-аналоговим перетворювачем (ЦАП, digital-to-analogue converter) [3].

Дискретне перетворення Фур'є (ДПФ) є базовим алгоритмом для цифрової обробки сигналів. Завдяки появі ефективних алгоритмів його обчислення (алгоритмів швидкого перетворення Фур'є) ДПФ широко використовується для цифрової фільтрації та спектрально-кореляційного аналізу сигналів.

Для дискретного сигналу пряме й обернене дискретне перетворення Фур'є має вигляд:

$$S(k)=\sum_{n=0}^{N-1} S(n)\exp\left[-j\frac{2\pi nk}{N}\right], k=0, \acute{N}-1; (1.3)$$

$$S(n)=\frac{1}{N}\sum_{k=0}^{N-1} S(k)\exp\left[j\frac{2\pi nk}{N}\right], n=0, \acute{N}-1, (1.4)$$

де k – номер гармоніки із частотою , N – обсяг вибірки. $S(k)$, визначений як комплексний спектр сигналу.

Дискретне перетворення Фур'є використовується для визначення гармонічного і частотного складу дискретних сигналів, а також дозволяє аналізувати, перетворювати і синтезувати сигнали, що неможливо за умови аналогової обробки [6].

Обчислення перетворень Фур'є в безпосередньому вигляді є дуже складним і містить дуже велику кількість операцій множення і обчислень синусів. Однак за допомогою алгоритму швидкого перетворення Фур’є складність обчислень суттєво зменшується. Цей алгоритм часто використовується для цифрової обробки сигналів і отримання частотних характеристик цифрового сигналу [7].

Існують такі методи синтезу звуку:

1. Адитивний (additive). Використовує твердження Фур’є про те, що будь який сигнал можна представити у вигляді суми синусоїдальних коливань з різними частотами та амплітудами. Для реалізації методу необхідний набір синусоїдальних генераторів з незалежним керуванням, вихідні сигнали яких сумуються. Перевагами методу є синтез довільного періодичного звуку і висока передбачуваність процесу синтезу. Однак звуки складної структури можуть вимагати великої кількості генераторів. Проблему вирішують заміною набору генераторів (реальних або математичних) зворотнім перетворенням Фур’є [6].

2. Різницевий (subtractive). По своїй сутності протилежний адитивному методу. В основу покладена генерація звукового сигналу з багатим спектром з наступною фільтрацією, яка полягає у виділенні одних складових і ослабленні інших. За таким принципом працює мовний апарат людини. Основними інструментами в цьому методі є керовані фільтри: резонансний (смуговий) з

налаштуванням положення і ширини смуги пропускання і фільтр нижніх частот із змінною частотою зріза. Також для кожного фільтра регулюється добротність (Q) - крутизна підйому або спаду на резонансній частоті [6].

Перевагами методу є досить широкий діапазон синтезованих звуків і доволі проста реалізація. Він застосовується у багатьох студійних та концертних синтезаторів, наприклад синтезатора Moog. Однак недоліком методу є потреба у великій кількості керованих фільтрів для виконання синтезу звуків із складним спектром [6].

3. Частотно-модуляційний (frequency modulation, FM). Використовує взаємну модуляцію по частоті між декількома синусоїдальними генераторами. Кожний з генераторів має власний формувач амплітудної огинаючої, амплітудний і частотний вібратор, який називається оператором. Алгоритм їх синтезу може включати один або більше операторів, з'єднаних послідовно, паралельно, послідовно-паралельно, із зворотними зв'язками та в інших сполученнях - все це дає велику множину можливих звуків [6]. Перевагою методу є велика різноманітність синтезованих звуків, які легко повторити на різних сумісних синтезаторах. Однак недоліком є те, що важко забезпечити приємний тембр в усьому діапазоні звучання, імітація звуку реальних інструментів досить груба, а також доволі складно організувати тонке керування операторами, тому зазвичай використовується спрощена схема з невеликим діапазоном можливих звучань [6].

4. Самплерний (sample – виборка, зразок). В цьому методі записується реальне звучання (сампл), яке відтворюється в потрібний момент. Для отримання звуку різної висоти відтворення семплу уповільнюється або прискорюється, а також застосовується розрахунок проміжних значень (інтерполяція). Для того щоб тембр звуку при зміні висоти не змінювався занадто сильно, використовують декілька семплів через певні інтервали (як правило, через одну-дві октави) [6].

Метод дозволяє отримати точну подібність звучання реального інструмента, однак для цього потрібні досить великі об'єми пам'яті. Крім того,

семпл звучить природньо тільки при тих параметрах, при яких він був записаний. Тому в сучасних синтезаторах використовується комбінація трьох основних методів синтезу, де в якості основного сигналу використовується семпл. Типовий представник цього класу синтезаторів – E-mu Proteus [6].

5. Таблично-хвильовий (wave table, WT). Різновид самплерного методу, в якому звучання записується не повністю, а тільки його окремі фази - атака, початкове затухання, середня фаза та кінцеве затухання, що дозволяє значно зменшити об’єм пам’яті, необхідний для збереження семплів. Ці фази записуються на різних частотах і при різних умовах (м’який або різкий удар по клавішах роялю, різне положення губ та язика при грі на саксофоні та ін.), в результаті чого отримують сімейство звучань одного інструмента. При відтворенні ці фази потрібним чином складаються, що дає можливість при відносно невеликому об’ємі самплів отримати достатньо широкий спектр різних звучань інструмента, а головне – помітно підсилити виразність звучання. Проблема цього методу в складності поєднання різних фаз суцільним звучанням і непомітними на слух переходами. Тому синтезатори цього класу досить рідкісні та дорогі [6].

6. Метод фізичного моделювання (physical modelling). Полягає в моделюванні фізичних процесів, що визначають звучання реального інструмента на основі його заданих параметрів. В зв’язку з високою складністю точного моделювання навіть простих інструментів і величезним об’ємом обчислень метод поки що розвивається повільно, на рівні студійних та експериментальних взірців синтезаторів. Очікується, що з момента свого достатнього розвитку він замінить відомі методи синтеза звучань акустичних інструментів, лишивши їм тільки задачу синтеза неприродніх тембрів. Прикладом сучасного використання цього методу є WaveGuide-технологія, що активно розробляється в Стенфордському університеті і застосовується вже в декількох промислових моделях електронних роялей, наприклад, фірми Baldwin [6].

З вище перерахованих методів синтезу звуку в звукових платах переважно використовуються таблично-хвильовий та частотно-модуляційний методи.

Методи обробки звуку:

1. Монтаж - полягає у вирізанні із запису одних ділянок, вставці інших, їх заміні, розмноженні та інше [6].

2. Амплітудні перетворення - операції над амплітудою сигналу, які в зводяться до множення значень амплітуд на постійний коефіцієнт (підсилення/ послаблення) або зміни в часі за допомогою функції-модулятора (амплітудна модуляція) [6].

3. Частотні (спектральні) перетворення - виконуються над частотними складовими звуку. При частотних перетвореннях потрібна обробка і наступна згортка за алгоритмом швидкого перетворення Фур'є, тому фільтрація в реальному часі поки що не реалізується на процесорах загального призначення. Для обробки у реальному часі використовуються сигнальні процесори (DSP) [6].

4. Фазові перетворення - зазвичай використовуються для зсуву фази сигналу або її модуляції за допомогою деякої функції або іншим сигналом. Слуховий апарат людини використовує фазу для визначення напрямку джерела звуку, тому фазові перетворення стереозвуку дозволяють отримати різні ефекти "об'ємності" звуку: ефект обертового звуку, хору, Surround та інші [6].

5. Часові перетворення - полягають у додаванні до основного сигналу його копій, зсунутих у часі на різні величини. При зсувах на величини, співрозмірні до величини періоду сигналу, такі перетворення стають фазовими. Якщо величина зсуву більша за період сигналу, то може бути отриманий ефект хору (chorus), який створює відчуття розмноження джерела сигналу (за умови зсуву менше 20 мс) або ефект реверберації (20 -50 мс) і відлуння (більше 50 мс) [6].

6. Формантні перетворення - є частковим випадком частотних і оперують з формантами - характерними смугами частот, які властиві людському голосу.

Кожному звуку відповідає своє співвідношення амплітуд і частот декількох формант, яке визначає тембр і розбірливість голосу. Змінюючи параметри формант, можна підкреслювати або послаблювати окремі звуки, міняти одну голосну на іншу, зсувати регістр голосу [6].

Лінійні фільтри поділяються на два великих класи за імпульсною перехідною функцією: фільтри з нескінченною (НІХ-фільтри) і скінченною (СІХ-фільтри) імпульсною характеристикою [9].

По виду частотної характеристики фільтри підрозділяються на:

- Фільтр низьких частот – пропускає низькі частоти сигналу (рис.1.7).
- Фільтр високих частот – пропускає високі частоти сигналу.
- Смуговий фільтр – пропускає обмежену смугу частот (рис.1.8).
- Режекторний фільтр пропускає всі частоти, крім певної смуги.
- Фазовий фільтр пропускає всі частоти сигналу, але змінює його фазу.

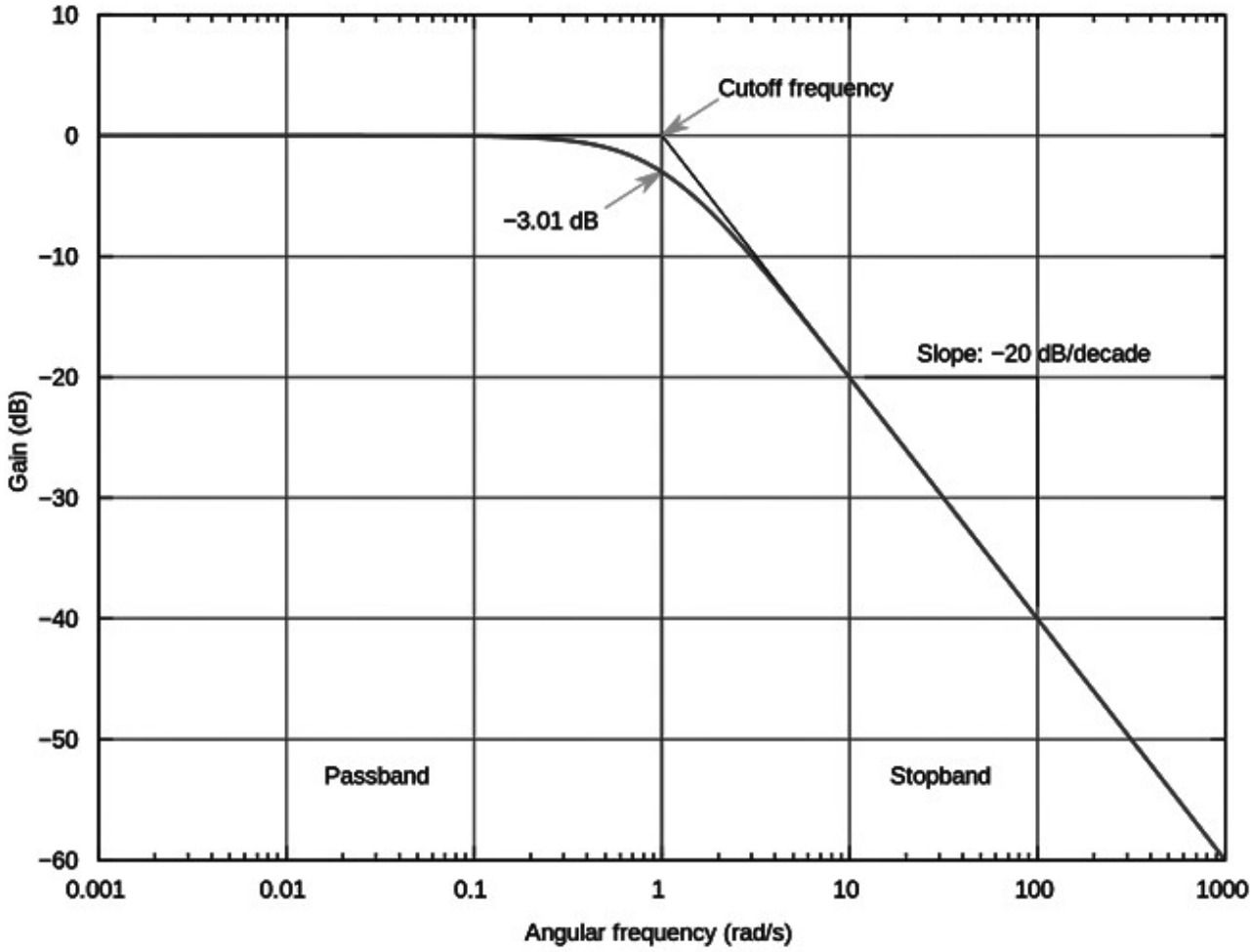


Рисунок 1.8 – АЧХ низькочастотного фільтру

Фільтр низьких частот – фільтр, який пропускає низькі частоти, та послаблює частоти, розташовані вище частоти зрізу фільтру.

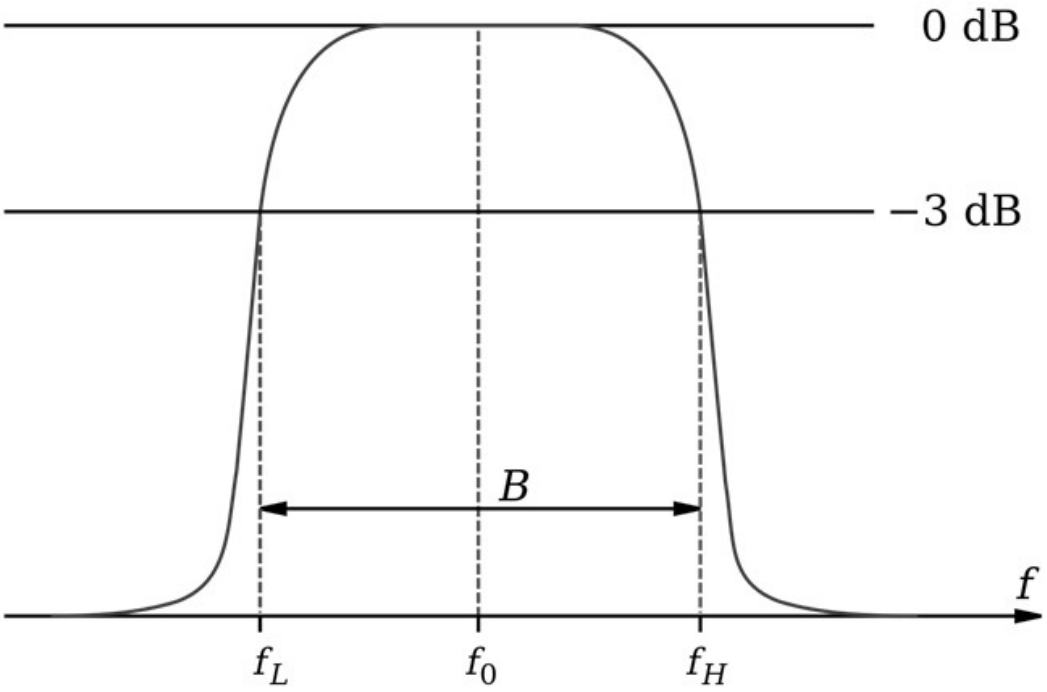


Рис. 1.8 – Смуговий фільтр

Смуговий фільтр – електронний фільтр, що пропускає сигнали в певному (діапазоні) (смузі) частот, і послаблює (вирізає) сигнали частот за межами цієї смуги. Наприклад, смуговий фільтр на 1800—1900 МГц пропускає тільки сигнали, частота яких лежить в інтервалі 1 800 ÷1 900 МГц.

1.4 Постановка задачі розробки

Оскільки завданням магістерської кваліфікаційної роботи є розробка програмної системи селективного аналізу і синтезу звукових сигнал, можна виділити такі задачі для виконання:

- дослідити тенденції розвитку технологій у сфері обробки звукових сигналів;
- проаналізувати сучасні тенденції у створенні веб програм для обробки звукових сигналів онлайн ;
- проаналізувати існуючі засоби для селективного аналізу і синтезу звукових сигналів;

- запропонувати покращення швидкодії селективного аналізу і синтезу звукових сигналів;
- розробити структуру інтерфейсу;
- створити концепцію програми;
- розробити програмний додаток;
- провести тестування додатку.

1.5 Висновки

В результаті аналізу стану проблеми було визначено, що проблема обробки цифрового звуку є досить актуальною, так як обробка звукової інформації все частіше впроваджується в сучасні в «хмарні сервіси», сформувавши один з пріоритетних напрямків в науці і виробництві сервісів для онлайн обробки та мастерінгу звукового контенту.

В ході проведення порівняльного аналізу існуючих аналогів було виявлено, що веб додатки мають досить обмежений функціонал у безпосередньому порівнянні з прикладними додатками. Також є проблеми з обробкою самих потоків (накладання різноманітних фільтрів для всього треку чи його частини та налаштування еквалайзеру) та використання одного потоку для обрахунків. Також суттєвим недоліком розглянутих існуючих реалізацій є те, що ці програмні продукти є платними, а їх безкоштовні версії мають досить обмежений функціонал та обмежені по часу використання. Був проведений аналіз алгоритмів і методів реалізації перетворення звукового сигналу в цифровий вигляд. Були розглянуті сучасні методи обробки звукового сигналу в браузері, і виявилось, що розробка програмної системи для селективного аналізу і синтезу звукових сигналів, з урахуванням всіх сучасних методів, а саме використання Web Audio API, потребує покращення.

2 РОЗРОБКА МЕТОДУ РЕАЛІЗАЦІЇ СЕЛЕКТИВНОГО АНАЛІЗУ ТА СИНТЕЗУ ЗВУКОВИХ СИГНАЛІВ

2.1 Вибір методів розв’язання поставленої задачі

На початку розвитку веб Internet explorer зробив спробу додати в браузер можливість додавання аудіо контенту, використовуючи тег <bgsound>,який дозволяв автоматично відтворювати midi файли при відкритті сайту. У відповідь на це розробники Netscape подібну функцію з тегом <embed> . Але жодне з цих рішень так і не було стандартизовано, і також не отримало підтримки від інших браузерів. Через декілька років в браузерах почали активно використовувати сторонні плагіни. Відтворювати аудіо стало можливим за допомогою Flash, Silverlight, QuickTime і інших плагінів. Вони добре виконували поставлені перед ними завдання, але все ж таки плагіни мали велику кількість недоліків. Тому можливість мати інструмент для роботи зі звуком, якій би був підтримуваний веб стандартами, вже довгий час потребувало рішення. З появою і масовим переходом на мобільні браузери, які не підтримують Flash, проблема стала ще більш нагальною.

Першим методом боротьби з «тишею в браузерах» без плагінів став елемент <audio>, який з’явився вже в першій специфікації html5. Він дозволяє відтворювати аудіо та стріми, контролювати відтворення, буферизацію і рівень звуку. Крім того, він є простий у розумінні і використанні. Наразі він є підтримуваним усіма браузерами, але не дає можливість вирішувати багато завдання пов’язаних з обробкою звукових сигналів.

Web audio API – потужний інструмент для маніпуляції звукової складової на веб-сторінці, що дає можливість розробникам вибрати джерела, можливість отримувати частоту, форму хвилі і інші дані з звукового джерела, додати до них спеціальні звукові ефекти (такі як panning), візуалізувати їх. Якщо порівняти можливості останніх двох методів, то отримаємо такий результат:

Завдання, які може вирішувати елемент <audio>:

- простий аудіо плеєр;
- однопоточне фонове аудіо;
- аудіо підказки, капчі.

Завдання, які може вирішувати Web audio API:

- об'ємний звук для ігор та інтерактивних веб додатків;
- додатки для обробки звуку;
- аудіо синтез;
- візуалізація аудіо.

Як бачимо з порівняння двох методів, використання Web audio API дає набагато більше можливостей для робити із звуковим сигналом та його обробкою. Даний метод дає можливість виконати синтез сигналу, зробити його аналіз, візуалізувати результат. Але даний метод має свої недоліки, наприклад:

1. API все ще перебуває на стадії чернетки.

2. Підтримка браузерами. IE і деякі мобільні браузери не підтримують використання даного методу.

3. Поки не існує універсального аудіо формату, який можна було б використовувати в веб додатках.

4. Відсутність багатопоточності.

Для реалізації програмного додатку було вирішено використовувати існуючі методи обробки звукових сигналів. Одними з найбільш поширених методів обробки звуку у браузері є Web Audio API.

Web audio API – потужний інструмент для маніпуляції звукової складової на веб-сторінці, що дає можливість розробникам вибрати джерела, можливість отримувати частоту, форму хвилі і інші дані з звукового джерела, додати до них спеціальні звукові ефекти (такі як panning), візуалізувати їх.

Web audio API дозволяє обробляти операції над аудіо за допомогою спеціального аудіо контексту (audio context), і був спроектований на використання модульної маршрутизації (modular routing). Базові операції виконуються за допомогою аудіо вузлів (audio nodes), що об'єднуються разом, формуючи аудіо-маршрутізаторную * таблицю (audio routing graph). Кілька джерел - з різними видами поточних схем - підтримуються навіть зсередини простого контексту. Ця модульна концепція забезпечує гнучкість в створенні складних функцій для динамічних ефектів [10].

Простий, типовий порядок дій виконання маніпуляцій над аудіо виглядає так:

- Створюється аудіо контекст;
- Всередині контексту визначаються джерела – такі як <audio>, генератор (oscillator), потік;
- Визначаються вузли ефектів, такі як реверберація (reverb), біквадратних фільтр (biquad filter), панорамування (panner), стиснення (compressor);
- Вибирається кінцева точка аудіо сигналу, наприклад системні звукові пристрої;
- Створюється зв’язок від джерела до ефектів, і ефекти до кінцевого сигналу.

У Web audio API є всього 28 інтерфейсів і відповідних подій, які згруповані в 9 категорій по функціональності [10].

1. Головні об'єкти визначення аудіо (AudioContext, AudioNode, AudioParam, ended (event))
2. Джерела звуку (OscillatorNode, AudioBuffer, AudioBufferSourceNode, MediaElementAudioSourceNode, MediaStreamAudioSourceNod)
3. Аудіо фільтри (BiquadFilterNode, ConvolverNode, DelayNode, DynamicsCompressorNode, GainNode, StereoPannerNode**,** WaveShaperNode, PeriodicWave)
4. Визначення звукових напрямків (AudioDestinationNode, MediaStreamAudioDestinationNode)
5. Аналіз і візуалізація даних (AnalyserNode)
6. Розщеплення та об'єднання аудіоканалів (ChannelSplitterNode, ChannelMergerNode)
7. Аудіо-просторовість (AudioListener**,** PannerNode)
8. Обробка аудіо через JavaScript (ScriptProcessorNode, audioprocess, AudioProcessingEvent)

9. Офлайн / фонова обробка аудіо (OfflineAudioContext, complete, OfflineAudioCompletionEvent).

Використовуючи можливості Web Audio API, при додаванні підтримки концепції потоків в рамках одного процесу, що називається багатопоточність, що в свою чергу дозволить використовувати паралельні багатопроцесорні методи обробки сигналів. Поєднавши вже існуючі реалізації обробки цифрових сигналів з реактивним багатопотоковим програмуванням, можна досягти підвищення швидкодії і оптимізації ефективності використання технічних засобів.

Які переваги використання багатопотоковості:

– Покращена реакція програми – будь-яка програма, що містить багато незалежних один від одного дій, може бути перепроектована так, щоб кожна дія виконувалося в окремому потоці. Наприклад, користувач інтерфейсу, що має паралельні потоки обробки, не повинен чекати завершення одного завдання, щоб почати виконання іншої.

– Більш ефективне використання багатопроцесування – як правило, додатки, що реалізують паралелізм через потоки, не повинні враховувати число доступних процесорів. Продуктивність додатки рівномірно збільшується при наявності додаткових процесорів. Чисельні алгоритми та програми з високим ступенем паралелізму, наприклад множення матриць, можуть виконуватися набагато швидше.

– Покращена структура програми – деякі програми більш ефективно представляються у вигляді декількох незалежних або напівавтономних одиниць, ніж у вигляді єдиної монолітної програми. Багатопоточні програми легше адаптувати до змін вимог користувача.

– Ефективне використання ресурсів системи. Програми, що використовують два або більше процесів, які мають доступ до загальних даних через пам'ять, що розділяється, містять більше одного потоку управління. При цьому кожен процес має повне адресний простір і стан в операційній системі.

Вартість створення і підтримки великої кількості службової інформації робить кожен процес більш витратним, ніж потік. Крім того, поділ роботи між процесами може зажадати від розробника значних зусиль, щоб забезпечити зв'язок між потоками в різних процесах або синхронізувати їх дії.

Реактивне програмування та спостережувані послідовності з RxJS.

Реактивний програмування – це програмування з асинхронними потоками (streams) даних. Або кажучи простими словами, реактивність – це здатність миттєво реагувати на будь-які зміни, в першу чергу це стосується реагування на зміну даних та орієнтацію на потоки. Якщо порівнювати два підходи до програмування, то на відміну від імперативного підходу, реактивний підхід будується на «push» стратегії поширення змін. «Push» стратегія реалізовує те, що в разі зміни даних ці самі зміни будуть «проштовхуватися», і залежні від них дані будуть автоматично оновлюватися.

Справа з асинхронною неблокуючою обробкою завжди була нормою у світі JavaScript, і зараз стає дуже популярною у багатьох інших контекстах.

Переваги зрозумілі: ефективне використання ресурсів. Але вигоди приносять свою ціну: нетривіальне збільшення складності. Реактивне програмування є складною за своєю природою, наш розум звик мислити послідовно.

Використання у веб програмуванні «Promises» (обіцянки) та «Futures» (майбутнє) спростили деякі найчастіші випадки, такі як "одноразові" асинхронні події, "запит зараз - відповідь пізніше", типовий для HTTP-запитів. Але якщо ми переходимо від подій "один раз" до "потоків подій", «Promises» (обіцянки) та «Futures» почнуть показувати деякі обмеження. У таких випадках ми можемо знайти ReactiveX і Observables як дуже потужний інструмент. Бібліотека ReactiveX потоків є майже для всіх мов програмування. Реактивні потоки варто використовувати, коли є потоки подій розтягнутий в часі (наприклад, для користувацького введення). Перевагою використання реактивних потоків є: підвищена швидкодія, більш ефективніше використання

ресурсів, функціонал, який написаний на реактивних потоках, можна легко доповнити чи розширити.

Для реалізації багатопотокової веб програми найкраще підійде такий потужний інструмент як RxJs, реалізація парадигми реактивного програмування для мови програмування JavaScript.

RxJS - це бібліотека для JS, яка використовує патерн Observable (з англ. "Оглядач") для спрощення обробки і компоновки асинхронного або callback коду [11]. Бібліотека надає основний тип Observable та кілька допоміжних типів (Observer, Schedulers, Subjects) і оператори роботи з подіями як з колекціями (map, filter, reduce, every і подібні з JavaScript Array).

Що означає потік в реактивному програмуванні (рис. 2.2)?

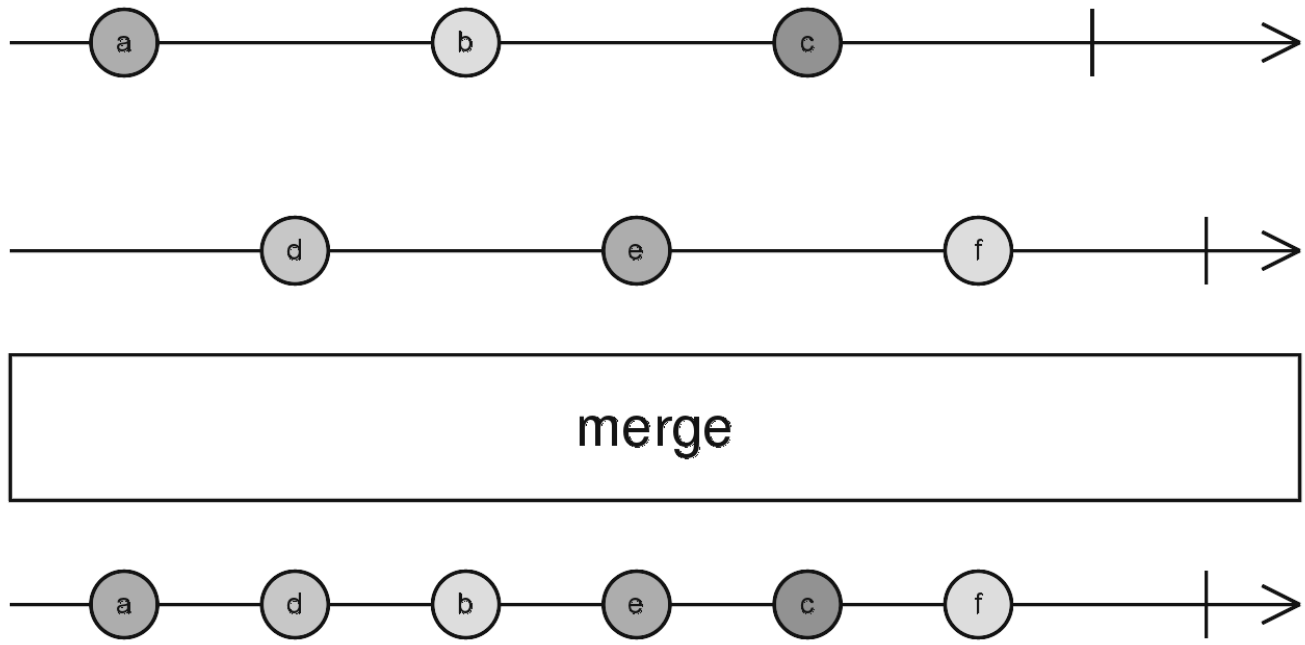


Рисунок 2.2 – Злиття двох потоків в RxJs

Потік – це масив даних, відсортованих за часом, який може повідомляти про те, що дані змінилися. І все що необхідно зробити це підписатися на потік і він автоматично повідомить коли відбудуться зміни. І це вміє робити бібліотека RxJs.

2.2 Розробка методу селективного аналізу

Найбільш часто використовуваними методами, заснованими на використанні спектральних характеристик шуму, є методи, що реалізують різні модифікації алгоритму вирахування амплітудних спектрів [20].

Як обґрунтування цих методів наводяться такі міркування.

Якщо стаціонарний сигнал $s(t)$, t = ... *-1,0,1,* .. зі спектральною щільністю потужності *Pss (iw)* спотворений адитивним стаціонарним шумом *n (t)* зі спектральною щільністю потужності *Pnn (iw),* який передбачається некорельованим з *x (t),* то спектральна щільність потужності перешкоди сигналу *x (t) - Pxx (iw)* дорівнює:

$$Pxx(iw)=Pss(iw)+Pnn(iw) \quad (2.1)$$

Отже спектральна щільність потужності корисного сигналу *s (n)* може бути оцінена як:

$$Pss(iw)=Pxx(iw)-Pnn(iw) \quad (2.2)$$

Отримана таким чином оцінка відповідає квадрату амплітудного спектра сигналу. відновлення мовного сигналу у часовій області здійснюється за допомогою зворотного перетворення Фур'є, причому фазовий спектр для відновленого сигналу береться таким же, як і у спостережуваного сигналу. У найбільш загальному вигляді операція спектрального віднімання може бути виражена співвідношенням:

$$|S(t,iw)|^2=\begin{Bmatrix} |X_i(t,iw)|^2, якщо |X_i(t,iw)|^2 \ge (A(t)+B|N(t,iw)|^2) \\ B|N(t,iw)|^2, інакше \end{Bmatrix} \quad (2.3)$$

Коефіцієнт A (t) (фактор переоцінювання), взагалі кажучи, залежить від співвідношення сигнал / шум на сегменті аналізу, І має типові значення близькі до 0.7 - 0.95, а коефіцієнт B (спектральний поріг) - вибирається в діапазоні 0.01 - 0.1.

Блок-схема алгоритму вирахування амплітудних спектрів приведена на наступному малюнку.

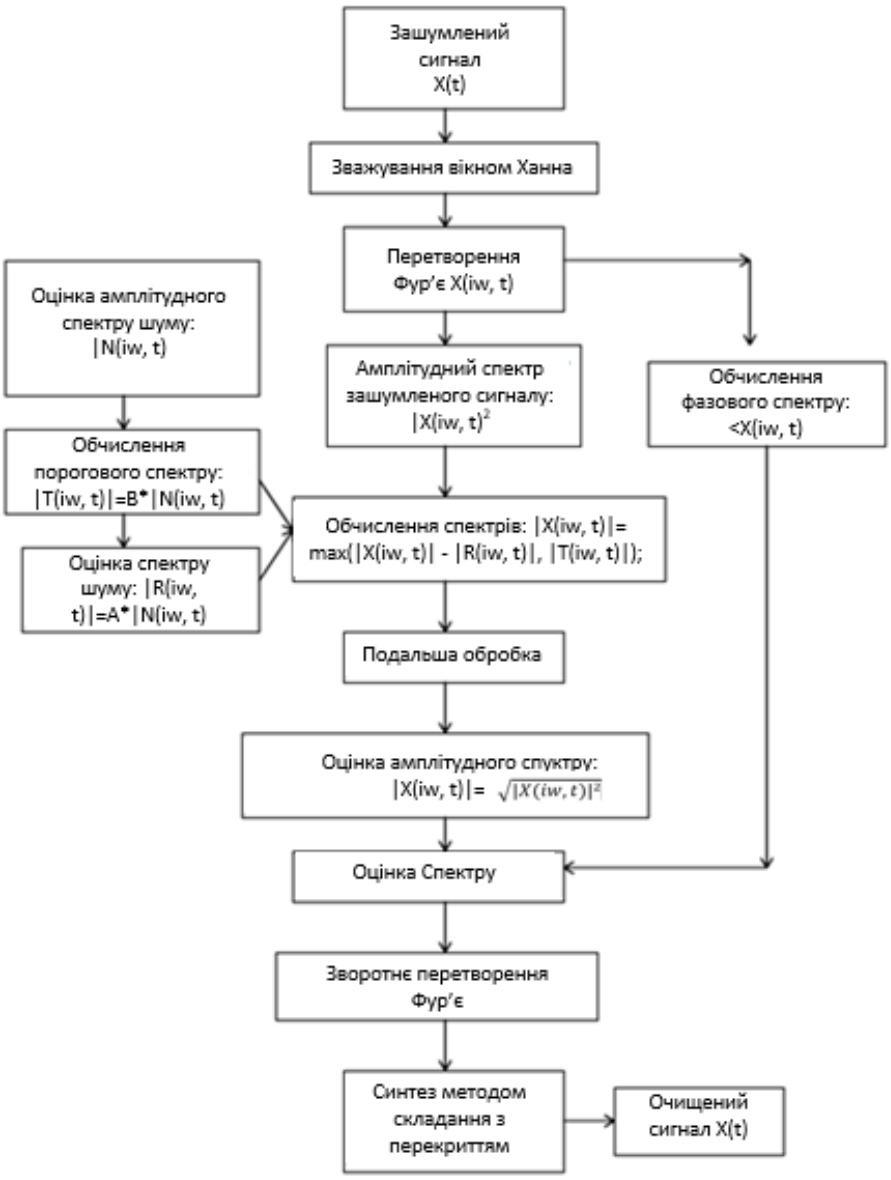


Рисунок 2.3 – Алгоритм обрахування амплітудних спектрів

У більшості випадків мова йде про пост-обробці сигналу в спектрально-тимчасової області. Ідея полягає в тому, що музичні тони в спектрально-

тимчасовому поданні є локальні (у часі і по частоті) спектральні максимуми, які, як правило, можна знайти з напів-евристичних міркувань [20].

Сценарій використання вирахування спектрів складається з трьох основних частин:

1. Вибору параметрів обробки (за потреби),
2. Виконання навчання на спектр шуму
3. Виконання процедури вирахування амплітудних спектрів.

При цьому є можливість регулювати процес обробки

2.3 Розробка методу синтезу звукових сигналів

Звуковий синтез це в першу чергу побудова спектрів звукових сигналів за допомогою генерації найпростіших форм хвиль і їх подальшої обробки. Цей процес дозволяє зімітувати звучання будь-якого інструменту, природнього звуку або навіть створити новий звук, який не існував до цього.

Будь-який синтезатор має кілька основних генераторів хвилі – осциляторів.

Осцилятор створює базову хвилю потрібної форми і тембру. Основними формами простої хвилі є: синусоїдна, пилкоподібна, прямокутна, трикутна.

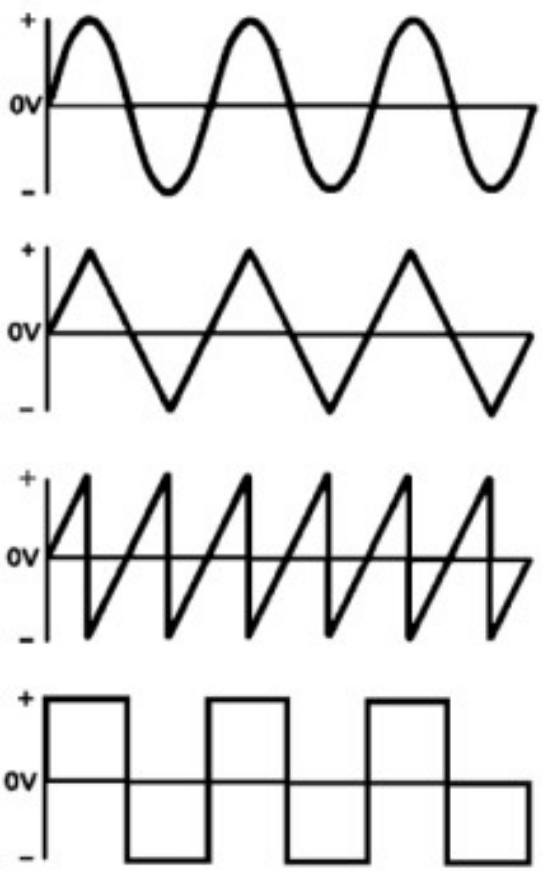


Рис. 2.4 – Основні форми простої хвилі

Змішуючи різні види хвиль з різних осциляторів можна істотно збагатити новими гармоніками спочатку простий звук, одиничний сигнал, а розкидаючи осцилятори по тональності можна домогтися ефекту звучання акорду.

Використовуючи поширені методи роботи зі звуком в браузері Web audio API, дає можливість обробляти операції над аудіо за допомогою спеціального аудіо контексту (audio context), який був спроектований на використання модульної маршрутизації (modular routing). Базові операції виконуються за допомогою аудіо вузлів (audio nodes), що об'єднуються разом, формуючи аудіо-маршрутизатор. Кілька джерел – з різними видами поточних схем підтримуються зсередини простого контексту (рис. 2.5).

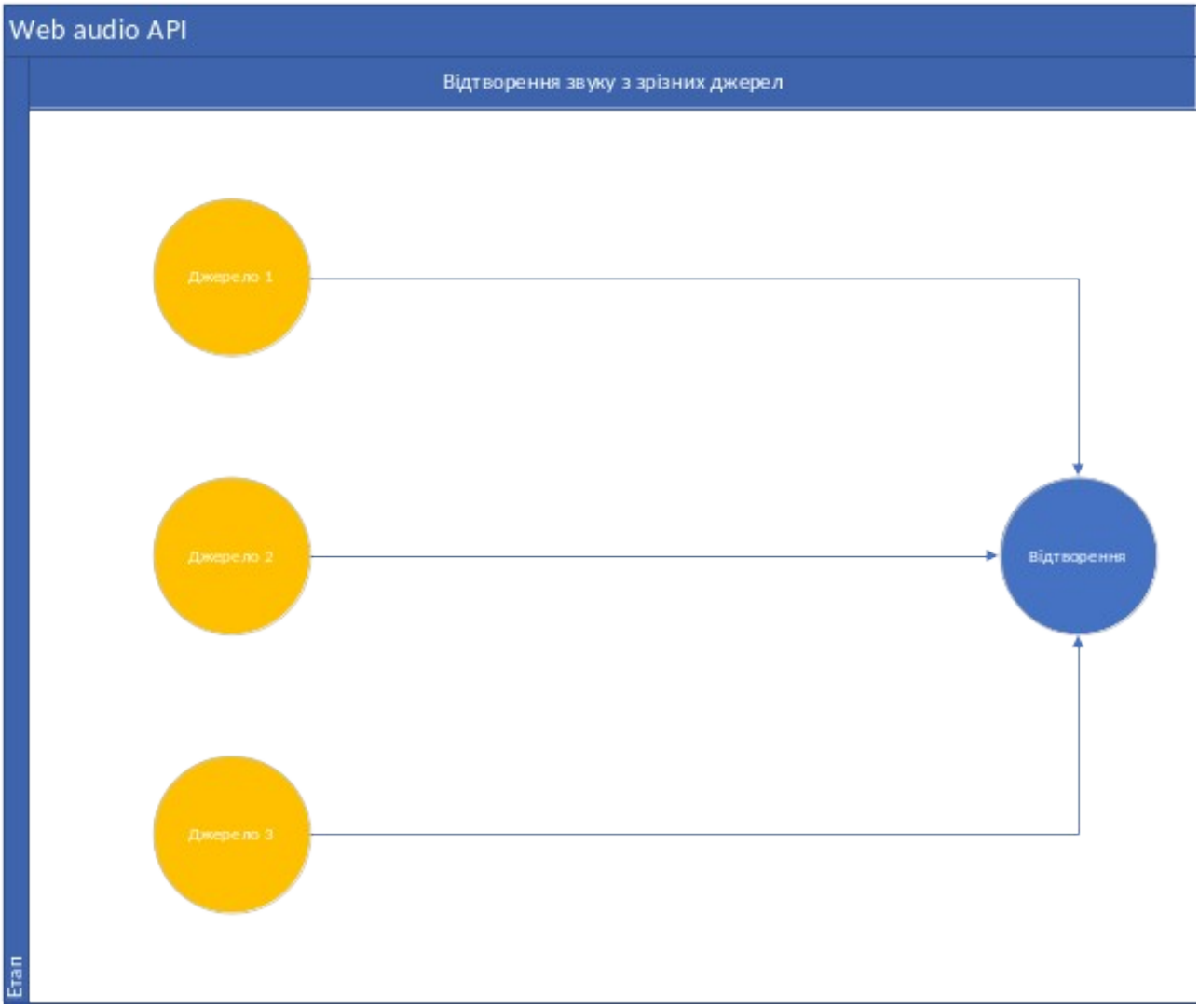


Рис. 2.5 – Відтворення звуку з різних джерел існуючими методами

Проте якщо розглядати механізм синтезу аудіо як конвеєр даних нот, які утворюються з різних джерел, таких як Midi або довільний сигнал, що передається через API, і закінчуючи генерацією тонів за допомогою Web Audio API. Запити на відтворення нот подаються у вигляді введених повідомлень об’єкту, що містить повідомлення, і потрапляють у загальний потік у нашому випадку «Subject» (тип «Observable»), який потім автоматично доставляє сигнали та інші інструкції службі, що їх відтворює. І все що необхідно зробити – це підписатися на даний потік даних. Загальна схема даного методу зображена на рисунку 2.6.

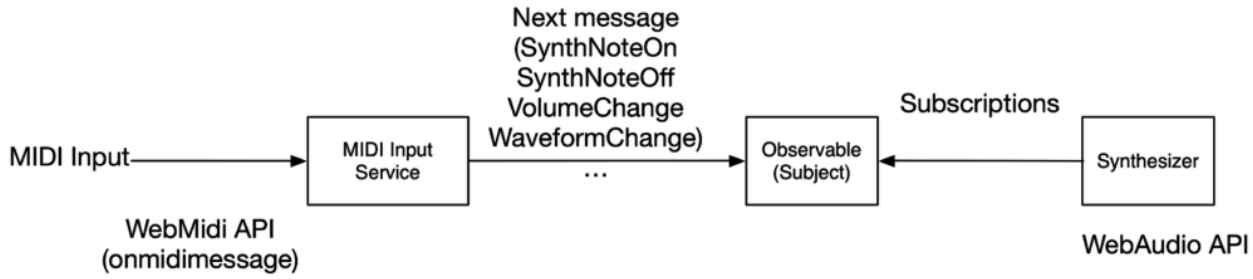


Рис. 2.6 – Метод виведення аудіо сигналів через потік

Схема відтворення звукових сигналів з використанням підходу реактивного програмування і використання потоків зображено на рисунку 2.7.

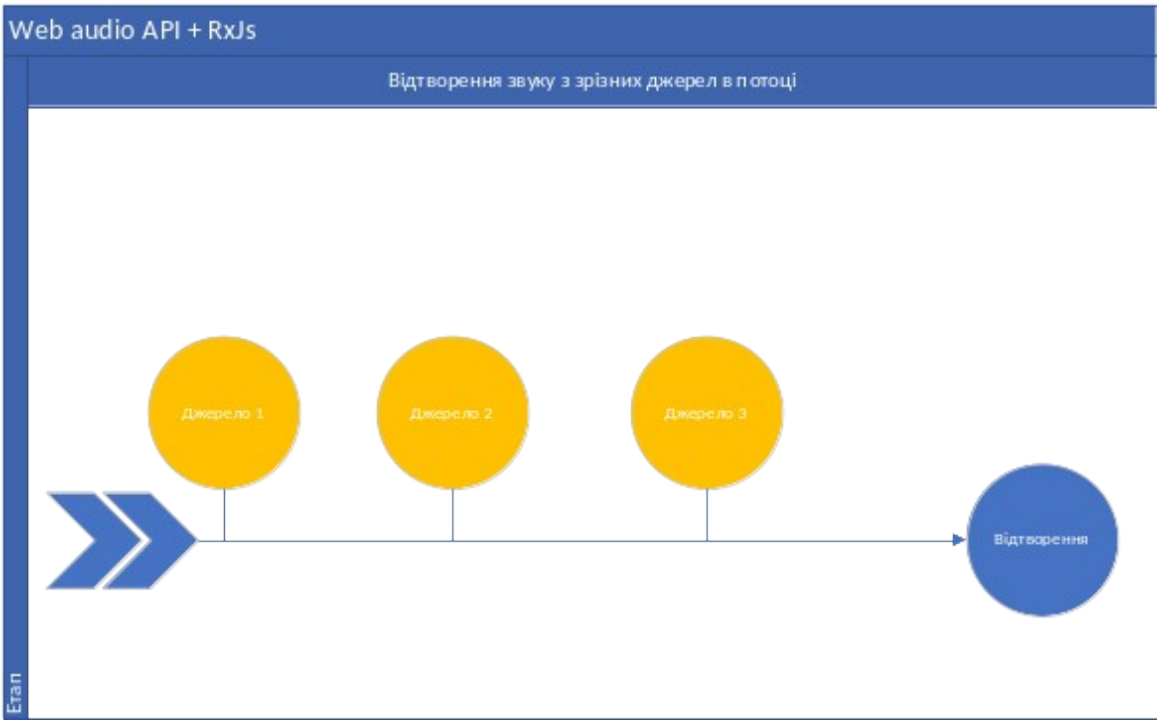


Рис. 2.7 – Схема роботи методу виведення аудіо сигналів через потік

Потоки дають нам можливість мати постійний потік інформації. Наприклад, з аудіо-передачі на сервері або з мікрофона на персональному комп’ютері (рис. 2.8).

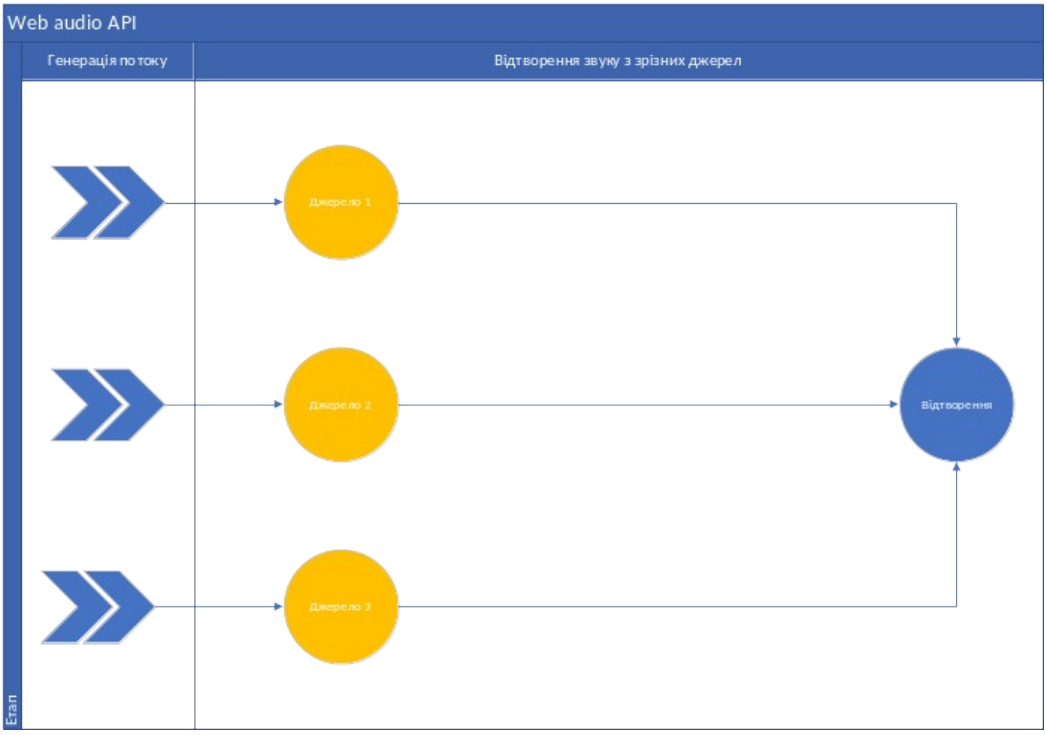


Рис. 2.8 – Виведення аудіо сигналів через декілька паралельних потоків

Простий, типовий порядок дій виконання маніпуляцій над аудіо виглядає так:

1. Створення аудіо контекст.
2. Всередині контексту визначаємо джерела - такі як <audio>, генератор (oscillator), потік.
3. Визначимо вузли ефектів, такі як реверберація (reverb), біквадратних фільтр (biquad filter), панорамміроване (panner), стиснення (compressor)
4. Виберемо кінцеву точку аудіо сигналу, наприклад системні звукові пристрої.
5. Об’єднуємо джерела до ефектів, і ефекти до кінцевого сигналу.

Створення звуку з генератора (осцилятора). В попередніх прикладах звуковий сигнал завантажували з ресурсів, але за допомогою осцилятора можливо створювати різні звуки на основі частоти. На рисунку 2.9 створюємо два осцилятори з різною частотою, які працюють в синхронно і без використання потоків.

Допоміжна функція createOscillator з WebContext повертає OscillatorNode, параметрами які ми можемо змінити з OscillatorNode – це частота періодичної форми хвиль у герцах і тип, що є формою періодичної форми хвилі.

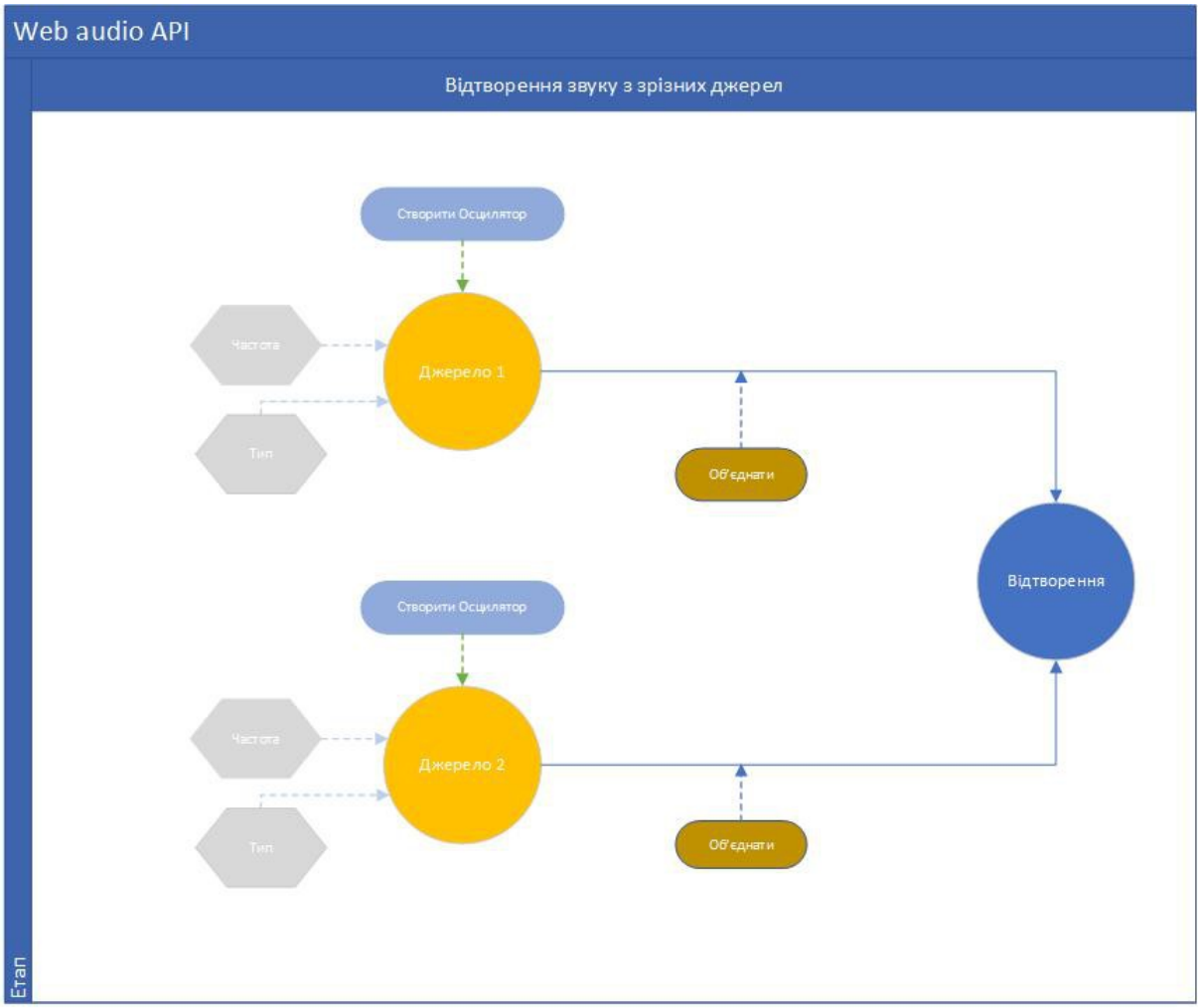


Рис. 2.9 – Створення двох звукових сигналів з різних джерел існуючими методами

Розглядаючи механізм створення звукових сигналів через генератор (осцилятор), як і у попередньому випадку, не просто як інформацію, що постійно змінюється, а як інформації з різних джерел, що додається до потоку (рис. 2.10). Такий метод підвищує швидкодію роботи програми, обробки та подальшого відтворення сигналу.

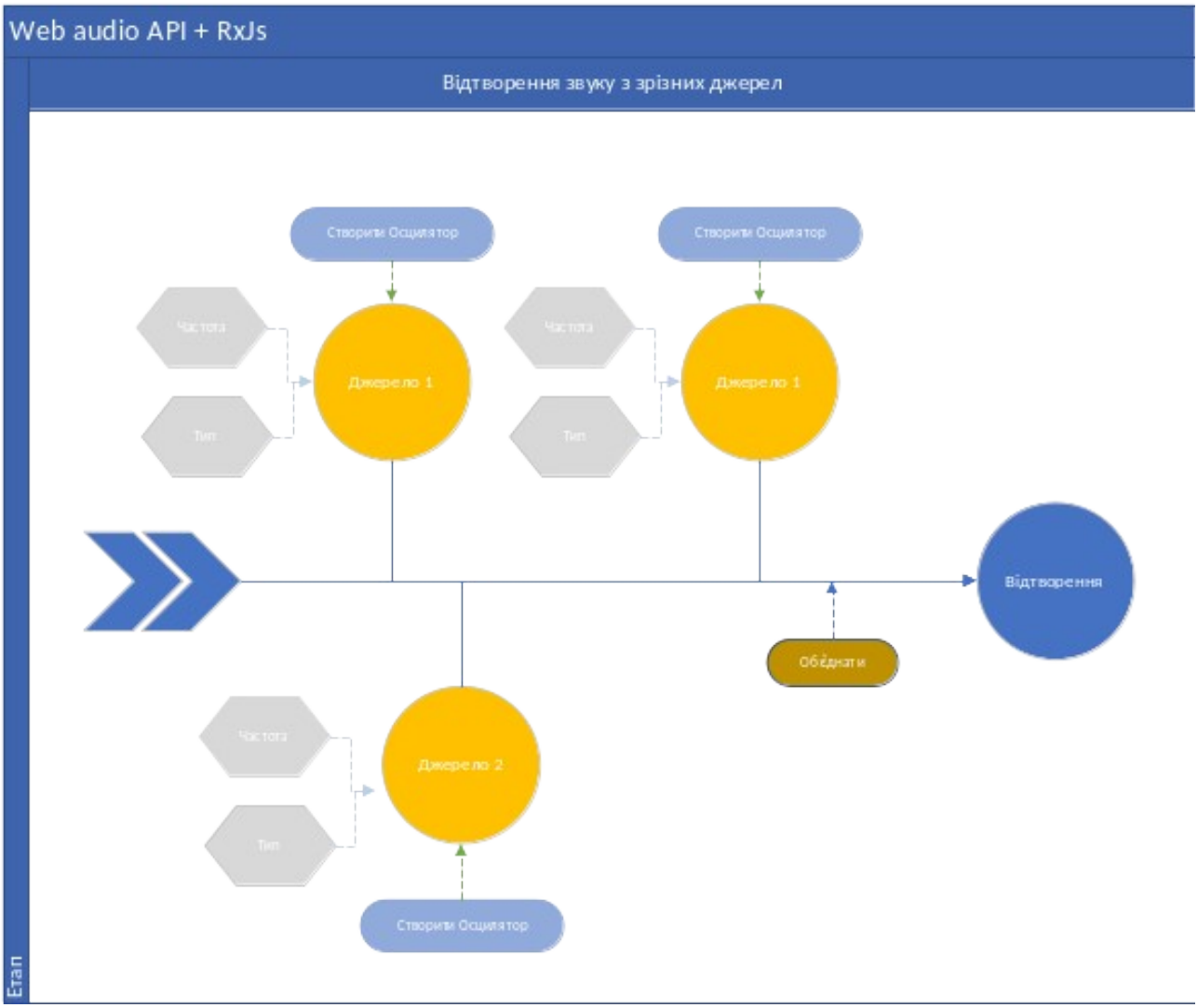


Рис. 2.9 – Створення трьох звукових сигналів з різних джерел через потік

Отже використання реактивного потокового методу створення та відтворення звукових сигналів підвищує швидкодію та оптимізує використання технічних засобів.

2.4 Вибір засобів програмної реалізації

В сфері веб-технологій процес розробки веб-додатків прийнято розподіляти на наступні етапи:

1. back-end development (розробка серверної частини веб-сайту);
2. front-end development (розробка клієнтської частини веб-сайту);
3. testing (тестування).

В подальшому відбувається процес Deployment (розгортання додатку на сервері) та Maintenance (супровід). Стек технологій, які слід використовувати, перш за все базується на основі вибору технологій та мови програмування для серверної частини сайту. На сьогоднішній день найпоширенішими серверними технологіями є:

1. PHP (Laravel, YII, Zend Framework, Symfony, CakePHP);
2. Java (Spring);
3. C# (.NET);
4. Ruby (Ruby on Rails);
5. Python (Django, Flask);
6. JavaScript (Node.js).

Розглянемо фреймворки для мови програмування JavaScript.

Node.js створений у 2009 році Райаном Далем. Він створив програмну платформу, засновану на JavaScript V8. Незвичним є те, що платформа має вбудовані бібліотеки для обробки запитів і відповідей, не потрібно використовувати сторонній веб-сервер і будь-які інші залежності. Node.js набирає обертів і використовується такими компаніями, як Microsoft, Yahoo, LinkedIn і PayPal.

Node.js характеризується такими властивостями:

- асинхронна однопотокова модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8.

Для керування модулями використовується пакетний менеджер npm (node package manager), що є найголовнішою перевагою, тому що npm є найбільшою у світі екосистемою бібліотек з відкритим кодом [12].

JavaScript – мова програмування, що використовується для додання до веб-сторінки певної інтерактивності та динаміки.

Одними з найбільш поширених фреймворків JavaScript є Angular, React та Vue.js. Порівняльний аналіз даних фреймворків був опублікований у тезах

Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії [2].

Angular представляє фреймворк від компанії Google для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків.

Angular часто називають MVW (Model-View-Whatever) фреймворк і серед головних переваг для стартапів є: швидке написання коду, швидке тестування будь-якої частини програми та двостороння прив'язка даних (зміни в back-end відразу ж відображаються на призначеному для користувача інтерфейсі).

Переваги Angular [13]:

- Підтримка веб-компонентів. Веб-компоненти Angular засновані на новому стандарті веб-компонентів, на відміну від закритої системи модулярізації AngularJS. На практиці це означає, що AngularJS дає можливість безпосередньо використовувати будь-який компонент, написаний як Web Component, не вдаючись до коду верстки.
- Використання Typescript. Найбільша комерційна перевага TypeScript полягає в його інструментарії. Ця мова дає можливості сучасного автозаповнення, навігації та рефакторінга. Такі інструменти стають практично незамінними при роботі з великими проектами.
- Відмінна продуктивність Angular не проводить глибокий порівняльний аналіз об'єктів. Якщо якийсь елемент додати в масив даних, зміну шляху не буде виявлено. Це стосується і властивостей об'єкта, поки вони не пов'язані безпосередньо з View.

React (старі назви: React.js, ReactJS) — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

Основна роль React – бібліотека для створення користувацьких інтерфейсів. Він фокусується на «View» частини MVC-розробки й дозволяє

створювати компоненти інтерфейсу, що зберігають свій стан. Це була одна з перших бібліотек, що реалізують віртуальне DOM-дерево. Статистика використання React може здатися досить низькою через те, що він використовується в основному в додатках, а не на сайтах.

Vue.js – JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних.

Vue.js реалізовує двосторонню прив'язку даних (як в AngularJS), візуалізацію на стороні сервера (як в Angular2 і ReactJS), Vue-cli (каркасний інструмент для швидкого старту) і опціональну підтримку JSX. Його творець стверджує, що Vue2 є одним із найшвидших фреймворків в цілому. Vue.js — це найкращий вибір для швидкої розробки крос-платформних додатків. Він може стати міцною основою для високопродуктивних додатків в одну сторінку (SPAs) і вигідним рішенням для тих випадків, коли продуктивність важливіше, ніж хороша організація коду або структура додатку.

Кожен із розглянутих фреймворків є досить потужним та має свої недоліки та переваги. Порівняльний аналіз був проведений із оцінюванням використання цих засобів для створення мережевого журналу, що дає можливість обміну персональними даними, зображеннями або мультимедійними файлами.

Vue.js є легким сучасним фреймворком, але є досить молодим проектом, що має менше ресурсів ніж в альтернативних варіантів, тому його використання є великим ризиком. React — це найбільш популярний фреймворк з найшвидшим розвитком, але складний для вивчення. React побудований на Javascript ES6, тоді як Vue – на ES5 або ES6, а Angular – на TypeScript. Використання в Angular TypeScript сприяло до підвищення надійності та безпеки, які необхідні для корпоративних додатків. Перевагою у сторону Angular є також те, що найбільш популярна бібліотека RxJS, що використовує реактивний підхід, була взята за основу цього фреймворка.

JetBrains WebStorm — інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину з його можливостей [14]. WebStorm постачається з наперед встановленими плагінами JavaScript (такими як для Node.js). WebStorm підтримує мови JavaScript, CoffeeScript, TypeScript та Dart.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладений Javascript) — в таких конструкціях підтримується коректний рефакторинг [15].

2.5 Висновки

Було проведено детальний аналіз існуючих методів реалізації поставленої задачі. Для програмної реалізації методів було обрано використовувати існуючий метод роботи з звуковим сигналом у веб-додатках Web audio API.

Під час виконання третього розділу було розроблено метод селективного аналізу звукових сигналів та метод синтезу звукових сигналів які будуть використовуватись під час реалізації програмного додатку.

Також, було проведено аналіз існуючих JS фреймворків та було обрано використовувати Angular для реалізації інтерфейсу користувача

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка загальної структури інтерфейсу

При створенні додатку було використано компонентну архітектуру Angular, яку можна порівняти з відомим архітектурним шаблоном MVC (Model – View – Controller). Відмінністю є те, що немає контролерів або ViewModels в Angular. Замість цього є компоненти, які складаються з шаблону, класів і метаданих (декораторів) [16].

Основні компоненти Angular, що зображені на рисунку 2.1:

- Модулі (Modules) – додатки Angular мають модульну структуру;
- Компоненти (Components) – керують відображенням даних в додатку;
- Шаблони (Templates) – шаблони відображення даних (шаблони в Angular схожі на звичайний HTML, проте мають синтаксичний «цукор» для зв'язку з компонентами);
- Метадані (Metadata) – для опису поведінки компонентів і класів;
- Data binding – зручний спосіб організації взаємодії шаблонів і компонентів;
- Директиви (Directives) – компоненти для розширення можливостей шаблонізатора;
- Сервіси (Services) – будь-які компоненти для забезпечення роботи логіки вашої програми;
- Впровадження залежностей (Dependency injection) – простий спосіб забезпечити екземпляри компонентів зовнішніми залежностями;
- Роутер (Router) – компонент для забезпечення навігації за додатком;
- Форми (Forms) – компонент для роботи з введеними користувачем даними;
- Анімація (Animations) – компоненти для анімації призначеного для користувача інтерфейсу.

Angular рекомендує використання модульного підходу, а саме модулів ES2015 які мають гарну підтримку у TypeScript, хоча це не є необхідністю. Використання TypeScript теж може бути замінено звичайним JavaScript, якщо необхідно, адже будь який TypeScript код компілюється у JavaScript. Використання модулів та TypeScript значно спрощує написання, структуризацію та підтримку коду тому відмовлятися від них не бажано. TypeScript є зворотньо сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js.

Компоненти контролюють певну частину DOM-елементів та містять логіку їх відображення й поведінки. Директиви, на відміну від компонентів, містять лише поведінку.

Рисунок 3.1 – Архітектура Angular для створення інтерфейсу користувача

Сервіси створені для зручного впровадження одного екземпляра класу в компонентах та директивах додатку, які від них залежать, тобто сервіси реалізують шаблон проектування «Одинак». Для позначення класу компонентом, директивою або сервісом Angular використовуються метадані, а саме функція декоратор, яка з необхідними параметрами розташовується перед класом.

Компоненти можуть використовувати інші компоненти як дочірні, складаючи зручну ієрархію елементів додатку. Зв’язування так само працюють й між батьківськими та дочірніми компонентами (рис. 2.2).

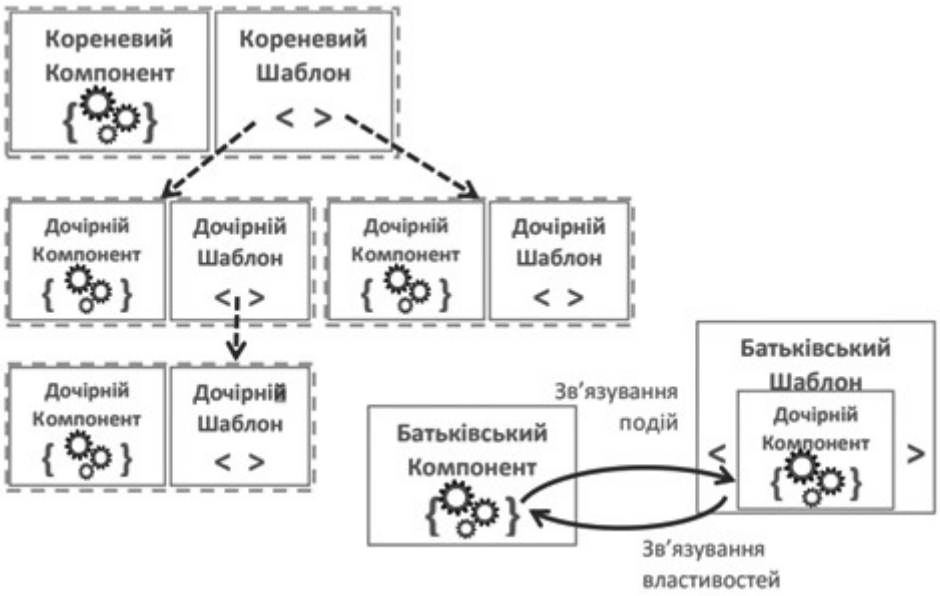


Рисунок 3.2 – Ієрархія та зв’язок компонентів Angular додатку

Розробка візуальної композиції графічного інтерфейсу користувача є важливою частиною розробки програмного додатку у сфері взаємодії між людиною та комп'ютером. Метою розробки є підвищення ефективності та легкості використання інтерфейсу програми. Інтерфейс користувача — це простір для обробки та відображення інформації, який максимально пристосований до потреб користувача [18].

Була побудована структурна схема інтерфейсу програмного додатку (рис. 3.3) і відповідних частин інтерфейсу (рис. 3.4, рис. 3.6, рис. 3.8).

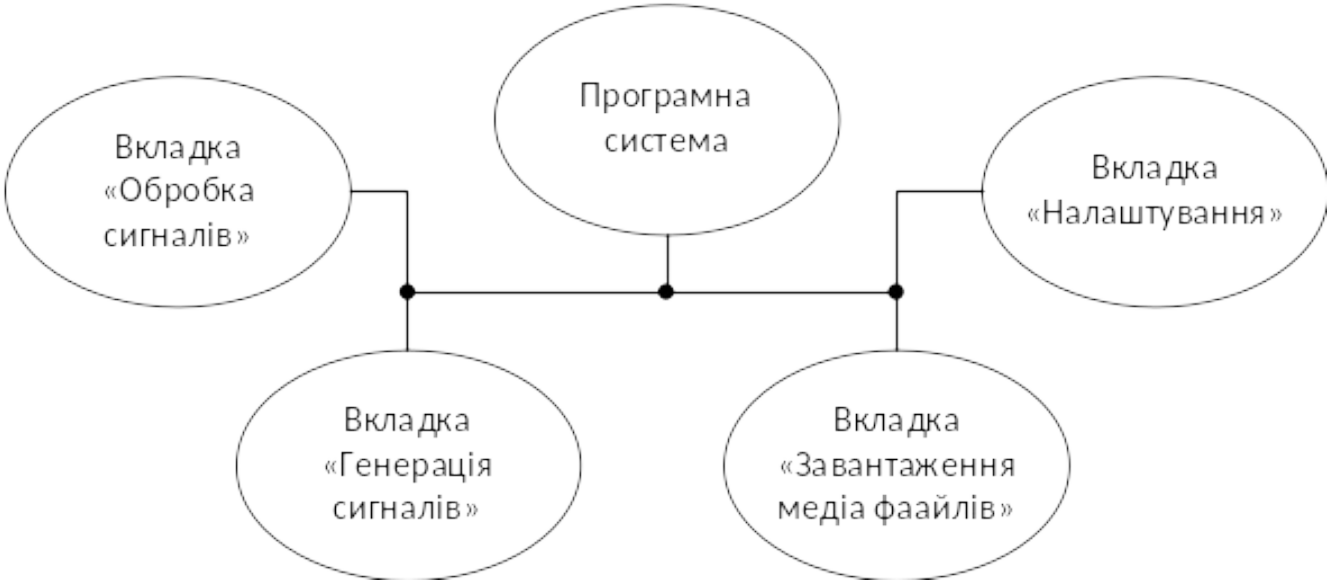


Рисунок 3.3 – Структурна схема інтерфейсу додатку

Розглянемо будову інтерфейсу користувача і програмну реалізацію кожної модуля.

3.2 Структура і програмна реалізація модуля завантаження аудіо-файлу

Однією з найбільш актуальних завдань для програм цифрової обробки є редагування вже існуючого аудіо-файлу, тому для виконання цього завдання потрібно реалізувати завантаження аудіо-файлу користувача на сервер для подальшої його обробки.

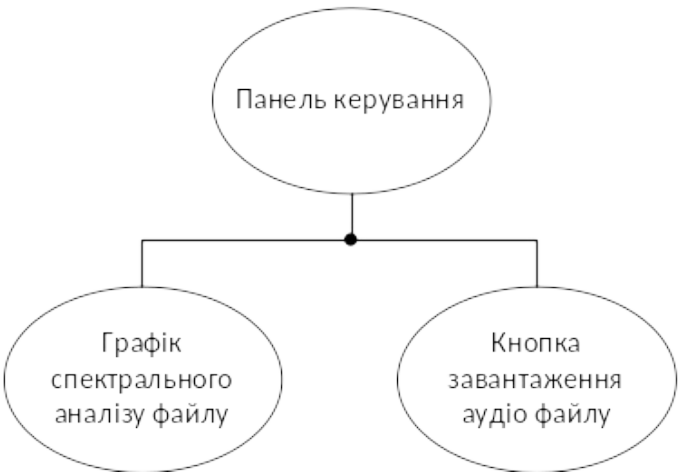


Рисунок 3.4 – Структурна схема інтерфейсу «Завантаження медіа файлів»

Для реалізації поставленого завдання використовувалась технологія Web Audio API, яка дозволяє програвати аудіо-файли форматів mp3, ogg або wav, що є цілком достатно для середньостатистичного користувача.

Для завантаження аудіо-файла на сервер і наступної його цифрової обробки необхідно виконати наступні операції:

1. Завантажити аудіо-файл.
2. Провести процедуру декодування завантаженого файлу.
3. Створити аудіо-буфер на сервері і вказати йому на декодований файл, який прийшов від клієнта.

   Описана вище процедура реалізована наступним чином:

```
window.AudioContext = window.AudioContext || window.webkitAudioContext;

play( snd ) {
        const audioCtx = new AudioContext();

        this.audioFileService.getFile().subscribe(request => {
    const audioData = request.response;

          audioCtx.decodeAudioData(
          audioData,
          buffer => {
                var smp = audioCtx.createBufferSource();
                smp.buffer = buffer;
                smp.connect( audioCtx.destination );
                smp.start( 0 );
          },
          err => this.store.dispatch(new BackendErrorMessage('Error with decoding audio data' + err.error)),
          );
        });

}

playFile(url) {
  play( url);
}
```

Рис.3.4 – Програмна реалізація завантаження аудіо-файлу на сервер

Для декодування аудіо-файлу використовується інтерфейс AudioContext і його функція decodeAudioData(), яка приймає в себе аудіо-файл, створює аудіо-буфер і визначає процедури наступної цифрової обробки.

Реалізація графічного інтерфейсу користувача для завантаження аудіофайлу на сервер виконана за допомогою Angular Materials, і виглядає наступним чином:

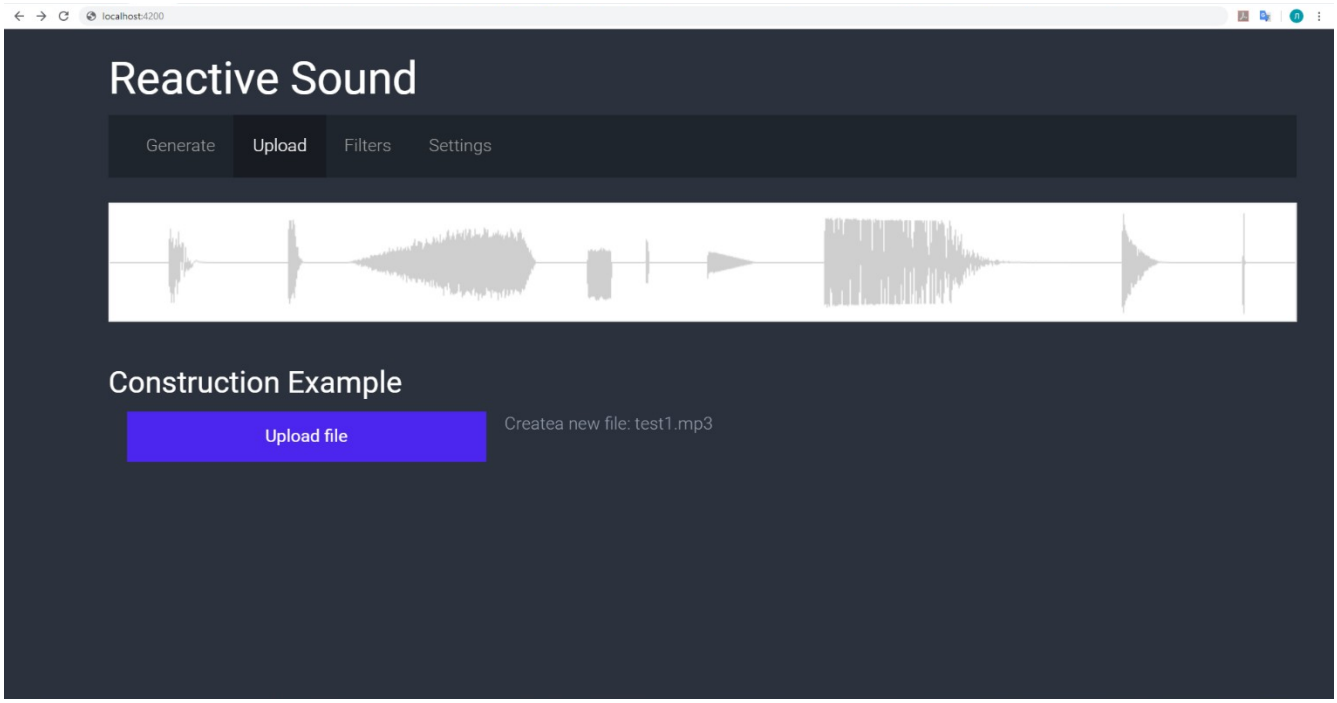


Рис. 3.5 – Графічна реалізація модуля завантаження аудіо-файлу

Отже, було завершено розробку програмного модулю для завантаження і побудови спектрального аналізу у вигляді графіку. Лістинг розроблених програмних модулів наведено в додатку Б.

3.3 Структура і програмна реалізація модуля генерації сигналів

Крім програвання аудіо-файлів, можливе генерування сигналів заданої частоти і форми. Технологія Web Audio API підтримує генерування синусоїдальних (sine), прямокутних (square), пилкоподібних (sawtooth), трикутних (triangle) імпульсів.

Створення генератора для аудіо-сигналів можливе за допомогою виклику функції context.createOscillator(), яка створює об’єкт класу OscillatorNode. Атрібутами цього класу є:

1. type – довзволяє задавати форму сигналу.
2. frequency – задає частоту сигналу.
3. detune – «тонке» налаштування частоти сигналу. Дозволяє змінити частоту сигналу в межах однієї октави (1200 сентів) вверх або вниз.

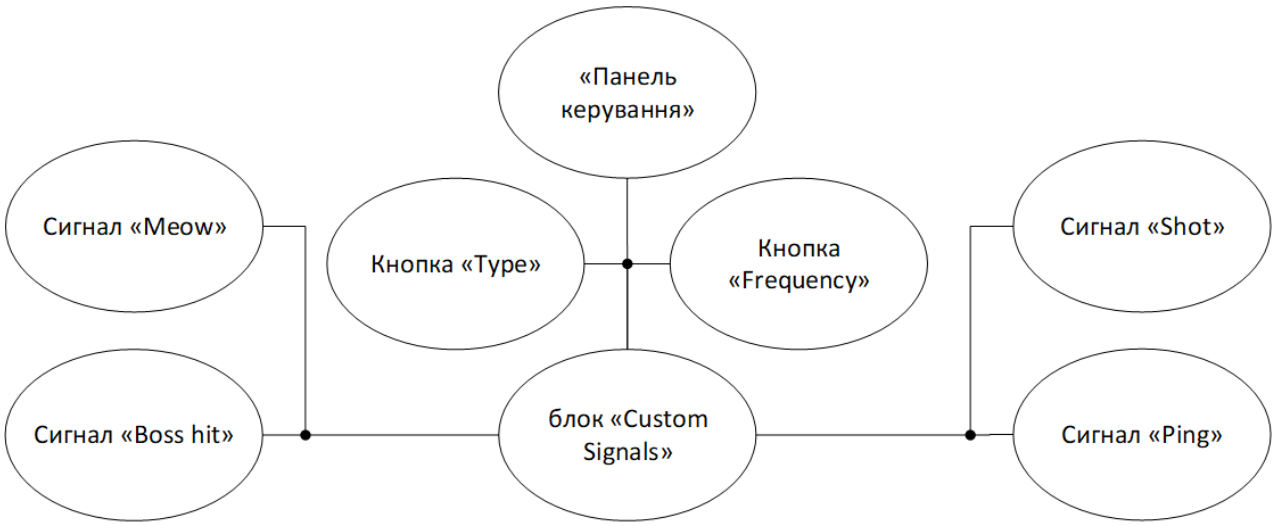


Рисунок 3.6 – Структурна схема інтерфейсу «Генерація сигналів»

Для того щоб створити об’єкти звукового сигнал, для початку необхідно створити сервіс SynthesizerService:

```
@Injectable()
export class SynthesizerService {
  // hold onto our notes once created
  private notes: Note[];
  setup(audioContext: Context, synthStream$: Subject<SynthMessage>,
        audioBus: AudioNode) {
    // start by setting static values for all notes
    Note.configure(audioContext, synthStream$, audioBus);
    // now, configure each note object
    this.notes = [
      new Note(['C0'], 16.3516),
      new Note(['C#0', 'Db0'], 17.3239),
      new Note(['D0'], 18.3540),
      new Note(['D#0', 'Eb0'], 19.4454),
      ...
      new Note(['D8'], 4698.64),
      new Note(['D#8', 'Eb8'], 4978.03),
      new Note(['E8'], 5274.041),
      new Note(['F8'], 5587.652)
    ];
  }
}
```

Рис.3.7 – Програмна реалізація сервісу генерації сигналу

Для запуску осцилятора створюємо новий генератор і обираємо вузол чутливості, запускаємо генератор, нарощуючи його до максимального обсягу, виходячи з часу атаки в секундах, (за замовчуванням у синтезаторі значення дорівнює 0). Найпростіший спосіб генерувати сингналу - це взаємодія з предметом спостереження (Subject Observable) , synthStream:

```
sendNote(note: number | string, duration: number = 300) {
    const self = this;
    this.synthStream$.next(new SynthNoteOn(note));
    setTimeout(() => {
      self.synthStream$.next(new SynthNoteOff(note));
    }, duration);
}
```

Рис.3.8 – Програмна реалізація створення нового осцилятора

Ось ключовий код у методі, який визивається при доставці повідомлення MIDI:

```
private processMusicNoteMessage(midiChannelMessage: any) {
    switch (midiChannelMessage.data[0]) {
    case 144:
        this.synthStream$.next(
            new SynthNoteOn(midiChannelMessage.data[1]));
        break;
    case 128:
        this.synthStream$.next(
            new SynthNoteOff(midiChannelMessage.data[1]));
        break;
    ...
    }
}
```

Рис.3.8 – Програмна реалізація методу генерації сигналу

Отже, надсилання повідомлення на предмет, що спостерігається, (Subject observable) просто виконується за допомогою методу «next()», що надсилає нове повідомлення, яке містить у собі характеристики сигналу, що дає можливість легко фільтрувати потрібні сигнали на іншому кінці потоку.

Графічний інтерфейсу користувача для генерації аудіо-сигналу виглядає наступним чином:

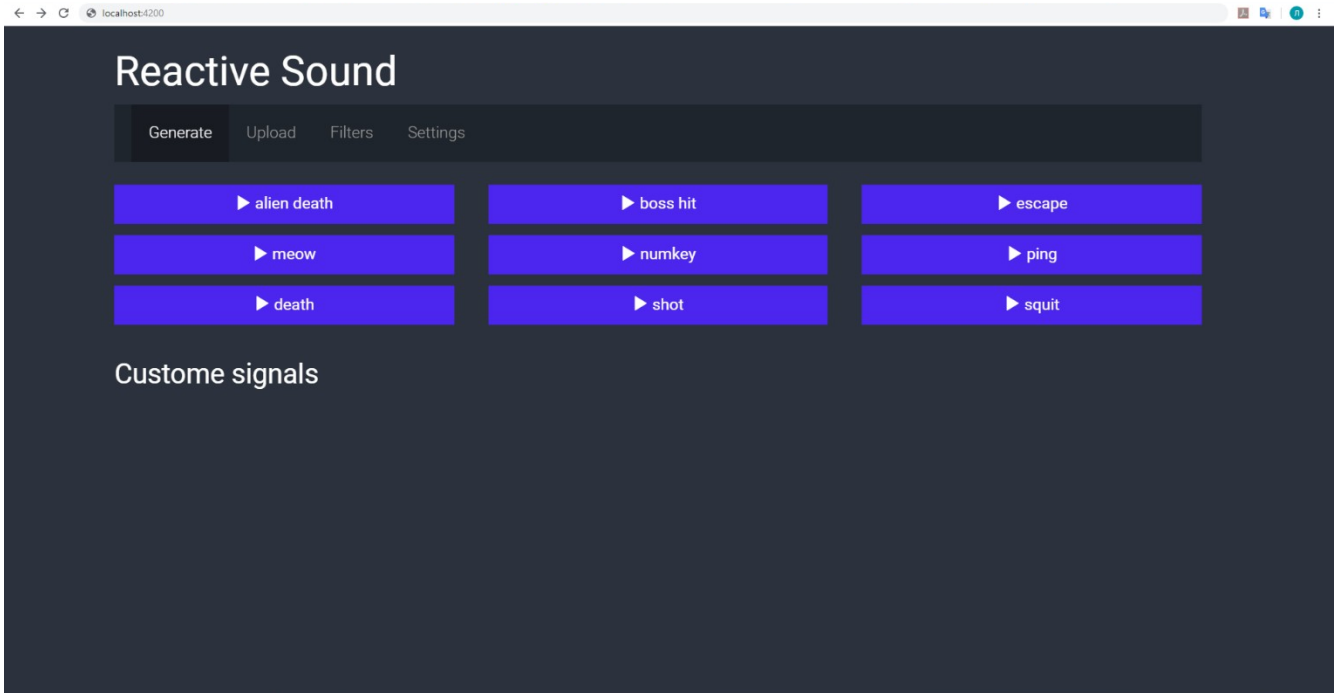


Рис. 3.9 – Графічна реалізація модуля генерації сигналів

Отже, було завершено розробку програмного модулю для генерації нового звукового сигналу. Лістинг решти розроблених програмних модулів наведено в додатку Б.

3.4 Структура і програмна реалізація модуля обробки сигналів.

Модуль обробки сигналів складається з таких логічних частин (рис. 3.4):

1. Відтворення або призупинення програвання аудіо-файлу.

2. Цифрова обробка аудіо-файлу, яка включає в себе гучність сигналу (gain), еквалайзер (equalizer), реверберація (reverb), стерео розведення каналів (stereo), спотворення (distortion) і фільтр «telephone».

3. Графічне відображення форми хвилі аудіо-сигналу або частотної характеристики.

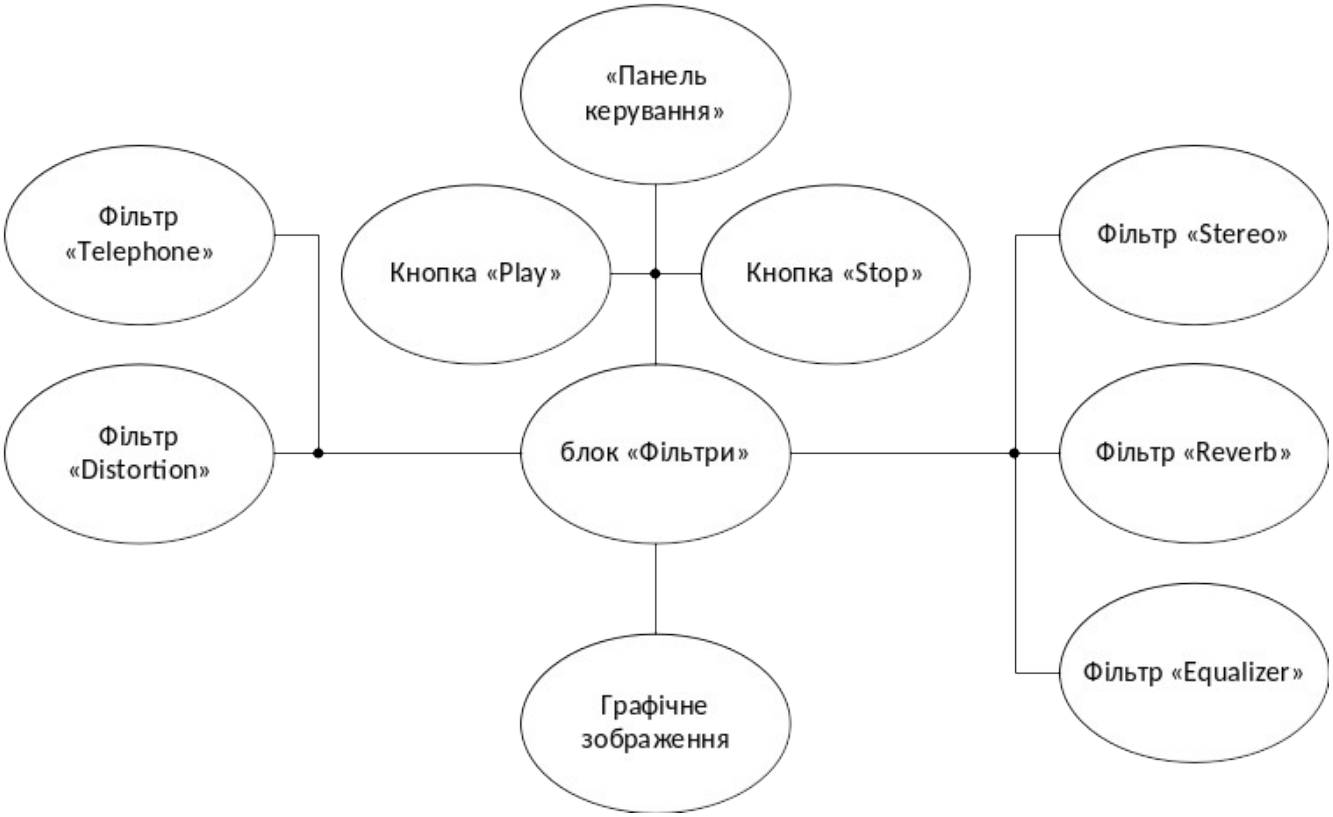


Рисунок 3.8 – Структурна схема інтерфейсу вкладки «Обробка сигналів»

Програвання аудіо-файлу реалізовується наступним чином.

Перше, що знадобиться - це створити обгортку для HTMLMediaElement. Об'єкт «source» – це перша ланка ланцюга, яку необхідно побудувати. У найпростішому випадку ланцюг складається тільки з двох ланок - джерело відразу підключається до виходу.

Єдиний, в даному випадку, параметр - це частота. Інші параметри збігаються для всіх фільтрів або змінюються під час роботи програми.

Type – тип фільтра. Може приймати одне із значень: lowpass, highpass, bandpass, lowshelf, highshelf, peaking, notch, allpass. Для прикладу використаєм

лише peaking фільтр – він дозволяє вибірково підкреслити або послабити обмежену смугу звукового спектра.

Q - добротність - змінює ширину смуги частот, на які фільтр впливає.

Gain – сила, з якою фільтр впливає на смугу частот.

Необхідно створити фільтри для всього набору частот. Для 10ти-смугового еквалайзера це можуть бути 60, 170, 310, 600, 1000, 3000, 6000, 12000, 14000 і 16000 Hz.

```
window.AudioContext = window.AudioContext || window.webkitAudioContext;

var context = new AudioContext(),
  audio = document.getElementById('audio');

var createFilter = function (frequency) {
  var filter = context.createBiquadFilter();

  filter.type = 'peaking'; // тип фильтра
  filter.frequency.value = frequency; // частота
  filter.Q.value = 1; // Q-factor
  filter.gain.value = 0;

  return filter;
};

var createFilters = function () {
  var frequencies = [60, 170, 310, 600, 1000, 3000, 6000, 12000, 14000, 16000],
    filters = frequencies.map(createFilter);

  filters.reduce(function (prev, curr) {
    prev.connect(curr);
    return curr;
  });

  return filters;
};

var equalize = function (audio) {
  var source = context.createMediaElementSource(audio),
    filters = createFilters();

  // источник цепляем к первому фильтру
  source.connect(filters[0]);
  // а последний фильтр - к выходу
  filters[filters.length - 1].connect(context.destination);
};

equalize(audio);
```

Рис.3.10 – Програмна реалізація фільтрів звукового сигналу

Графічний інтерфейсу користувача для синтезу і фільтрації аудіо-сигналу виглядає наступним чином:

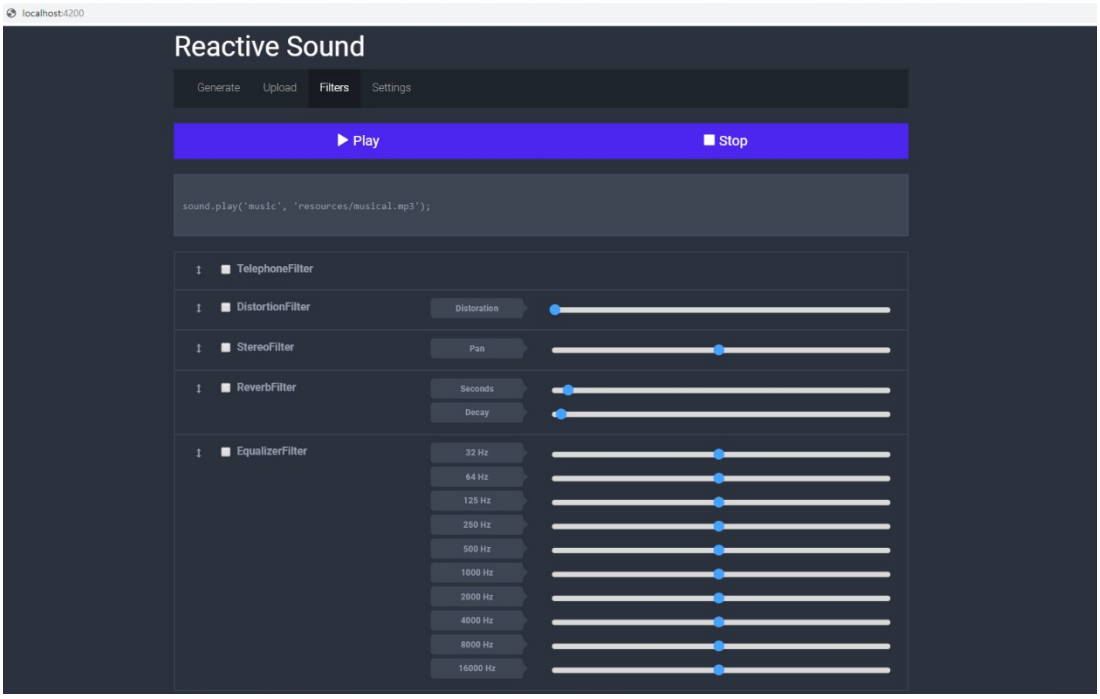


Рисунок 3.9 – Графічна реалізація модуля обробки сигналів

Отже, було завершено розробку програмного модулю для обробки звукового сигналу. Лістинг решти розроблених програмних модулів наведено в додатку Б.

3.5 Висновки

При створенні додатку було використано компонентну архітектуру Angular. Також, було розроблено структуру структурну схему додатку та графічні схеми інтерфейсу головних вікон. Було розроблено програмні модулі, які забезпечують роботу програмного додатку та реалізовують запропоновані раніше методи.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Вибір методики тестування

Тестування програмного забезпечення - це дослідження, проведене для надання зацікавленим сторонам інформації про якість випробуваного програмного продукту чи послуги. Тестування програмного забезпечення також може забезпечити об'єктивний, незалежний перегляд програмного забезпечення, що дозволяє підприємству оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають в себе процес виконання програми або програми з метою виявлення програмних помилок (помилок або інших дефектів) та перевірки того, що програмний продукт підходить для використання. Програмний продукт може оцінюватись за такими критеріями [17]:

- відповідність вимогам, якими керувалися проектувальники та розробники;
- правильна відповідь для усіх можливих вхідних даних;
- виконання функцій за прийнятний час;
- практичність;
- сумісність з програмним забезпеченням та операційними системами;
- відповідність задачам замовника.

Основна мета тестування полягає у виявленні несправностей програмного забезпечення, з тим щоб дефекти могли бути виявлені та виправлені. Тестування не може встановити, що продукт працює належним чином за будь-яких умов, але може встановити лише те, що він не працює належним чином за певних умов.

Сфера тестування програмного забезпечення часто включає вивчення коду, а також виконання цього коду в різних середовищах та умовах, вивчення аспектів коду: чи робить він те, що він повинен робити, і не робити те, що не

потрібно робити. У сучасній культурі розробки програмного забезпечення організація тестування може бути відділена від команди розробників. Існують різні ролі для тестування членів команди. Інформація, отримана з тестування програмного забезпечення, може бути використана для виправлення процесу розробки програмного забезпечення [18].

Загальний підхід до розробки програмного забезпечення часто визначає, коли і як проводиться тестування. Наприклад, в поетапному процесі більшість випробувань відбувається після того, як системні вимоги були визначені, а потім реалізовані в тестованих програмах. На відміну від того, під Agile підхід, вимоги, програмування та тестування часто виконуються одночасно [19].

Найбільш відомі методи тестування це тестування «білої скриньки» та тестування «чорної скриньки».

Тестування «білої скрині» - це метод тестування програмного забезпечення, який перевіряє внутрішні структури додатка (коректність побудови елементів додатку та правильність їх взаємодії), а не функціональность (на відміну від тестування «чорного ящика»).

Тестування чорної скриньки розглядає програмне забезпечення як «чорну скриню», поведінку якого можна визначити лише аналізом значень на його входах та відповідних виходах, вивчаючи функціональні можливості без будь-яких знань про внутрішню реалізацію. Тестувальники знають лише про те, що повинно робити програмне забезпечення, а не про те, як це робиться [18].

4.2 Тестування програми

Для перевірки коректності роботи додатку було проведене тестування, що складалося з тестування завантаження, генерації сигналу та тестування обробки звукових сигналів.

Тестування обробки зображення складалося з наступних дій:

1. Запуск програми (рис. 4.1). При завантаженні додатку автоматично завантажується перша вкладка «Generate», що містить у собі засоби генерації звукових сигналів.

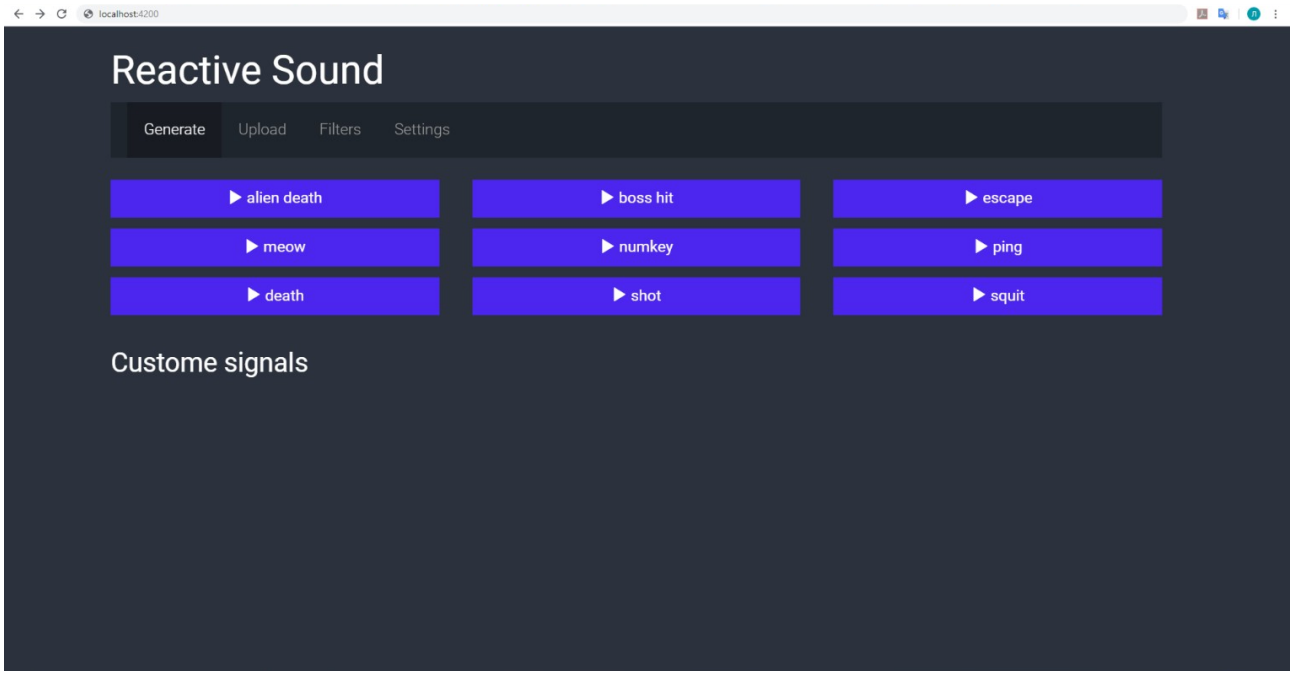


Рисунок 4.1 – Процес генерації звукових сигналів

Результат тестування генерації сигналів є успішний, було створено і відтворено усі можливі сигнали.

При натисненні пункту меню «Upload», відкривається відповідна вкладка, що містить у собі дві області: спектрального аналізу та завантаження файлу.

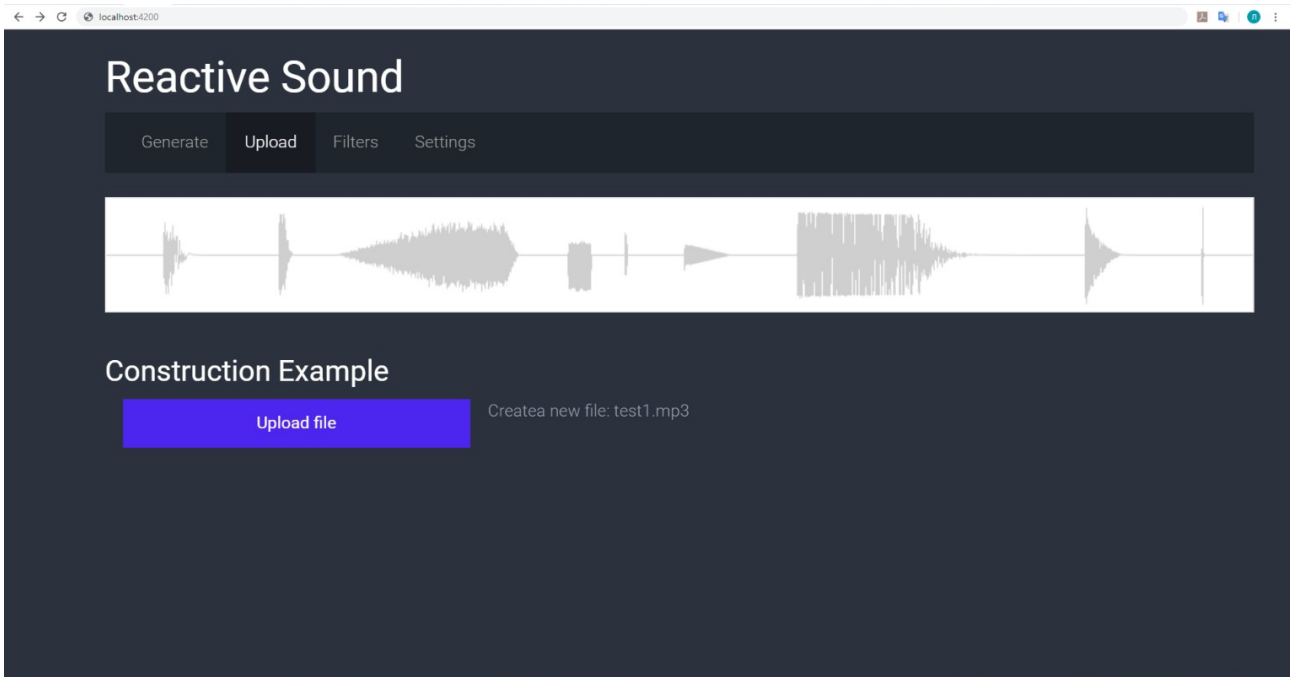


Рисунок 4.2 – Процес завантаження і аналізу аудіо файлу

Результат тестування завантаження файлу та спектрального аналізу є успішний.

При натисненні пункту меню «Filters», відкривається відповідна вкладка. Головними засобами для керування обробкою звуку у вкладці «Filters» є кнопка «Play» та кнопка «Stop». Також є списки можливих для застосування фільтрів, ефектів та еквалайзер.

Список доступних фільтрів:

1. «TelephoneFilter» - ефект звуку з телефону.
2. «DistortionFilter» - ефект «перегруженого» спотвореного сигналу.
3. «StereoFilter» - ефект стерео сигналу.
4. «ReverbFilter» - ефект реверберації.
5. «EqualizerFilter» - 10 смуговий еквалайзер.

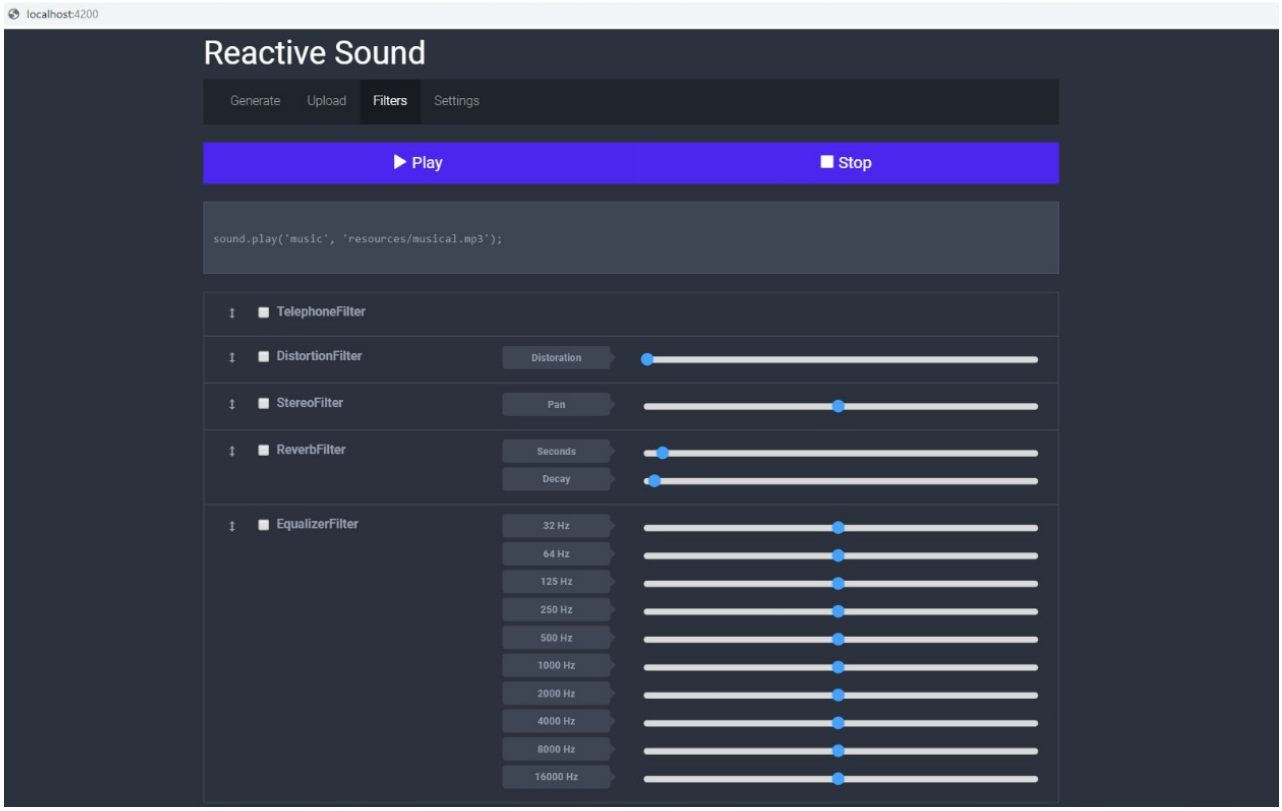


Рисунок 4.3 – Процес обробки аудіо файлу

Результат тестування відтворення і селективна обробка файлу є успішний.

4.3 Висновки

Було проаналізовано основні методи тестування: тестування «чорної скриньки» та тестування «білої скриньки». Для тестування програмного додатку було обраного стратегію «чорної скриньки». Так як воно забезпечує повну перевірку всіх функціональних вимог до програми. Було описано методику тестування програмного додатку і результати тестування програми. Тестування пройшло успішно, результати відповідають вимогам.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами: Рейда Олександр Миколайович (к.т.н., доцент каф. ПЗ, ВНТУ), Черноволик Галина Олександрівна (к.т.н., доцент каф. ПЗ, ВНТУ).

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями, що наведені в таблиці 5.1 [21].

Таблиця 5.1

Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.Термін окупності 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років.Термін окупності 5-ти років	Термін реалізації ідеї менше 3-х років.Термін окупності від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів	Необхідно отримання великої кількості дозвільних документів, що вимагає значних коштів та часу	Процедура отримання дозвільних документів вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.2.

Таблиця 5.2

Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Рейда	2. Черноволик
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	3	4
4	4	3
5	3	3
6	4	4
7	4	3
8	4	4
9	3	3
10	4	3

Продовження таблиці 5.2

11	3	4
12	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 42$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 42{,}5$	

Отже, з отриманих даних таблиці 5.2 видно, що нова розробка має високий рівень комерційного потенціалу. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 5.3.

Таблиця 5.3

Рівні комерційного потенціалу розробки

Середньоарифметична сума балів , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 42,5 балів, що відповідає рівню «вище середнього».

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи.

Проведемо прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько–технологічної роботи для розробки програмного забезпечення.

Виконаємо розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи, за такими статтями та формулами, приймаючи до уваги те, що для розробки інформаційної технології було залучено одного розробника програмного забезпечення.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \qquad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 22 дні;

t - число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.4.

Таблиця 5.4

Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	5800	263,63	7	1845,41
Інженер-програміст	3800	172,72	50	8636
Всього:				10481,41

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 10481,41 = 1048,14 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп} = (З_о + З_р) \cdot \frac{\beta}{100}, \quad (5.2)$$

$$Н_{зп} = (10481{,}41 + 1048{,}14) \cdot \frac{36{,}3}{100} = 4185{,}22 \, (грн.).$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.5

Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
Персональний комп'ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.6

Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	170	1	170
Пачка паперу	уп.	120	1	120
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				330

Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп'ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф=195 год.);

$K_п$ – коефіцієнт використання потужності ($K_п < 1$, $K_п = 0{,}8$).

$$В_е = 1{,}7 \cdot 0{,}6 \cdot 195 \cdot 0{,}8 = 159{,}12 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$B_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$B_{ін} = 1 * (10481,41 + 1048,14) = 11529,55 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$B = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 10481,41+1048,14+4185,22+159,12+625+330+11529,55= 28358,44 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$B_{заг} = \frac{B_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$B_{заг} = \frac{28358,44}{1} = 28358,44$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{B_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{28358,44}{0,9} = 31509,37 \text{ (грн.)}$$

Отже, прогноз загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи складає 31509,37 (грн).

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки.

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства Δ $П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_{1}^{n} ¿¿ \qquad (5.9)$$

де Δ $П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

Δ N – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 15 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 15 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 300 користувачів, протягом другого року – на 250 користувачів, протягом третього року – 150 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 600 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 250 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 15 \cdot 600 + (250 + 15) \cdot 300 = 88500 \, грн.$$

Протягом другого року:

$$\Delta П_2 = 15 \cdot 600 + (250 + 15) \cdot (300 + 200) = 154750 \, грн.$$

Протягом третього року:

$$\Delta П_3 = 15 \cdot 600 + (250 + 15) \cdot (300 + 250 + 150) = 194500 \, грн.$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $E_{абс}$ вкладених інвестицій розраховується за формулою:

$$E_{абс} = (ПП - PV), \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

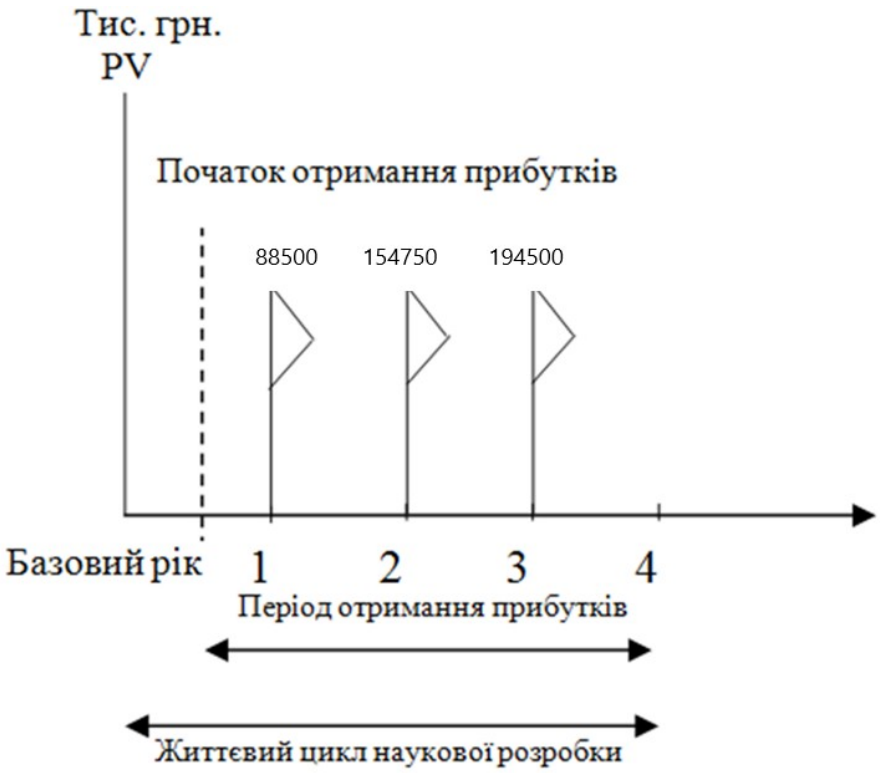


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_{1}^{m} \frac{\Delta П_i}{¿¿}¿ \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{31509,37}{¿¿} (\text{грн.})$$

Тоді розрахуємо $E_{абс}$:

$$E_{абс} = 353761,93 - 31509,37 = 322252,56 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$ за формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1 \quad [\text{грн}], \qquad (5.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{322252,56}{31509,37}} - 1 = 1,23 \text{або } 123\ \%$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3$$

Оскільки $E_в$ = 123% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}$$

$$T_{ок} = \frac{1}{1,23} = 0,81\ року$$

Термін окупності вкладених у реалізацію проекту інвестицій становить 0,81 року. Оскільки $T_{ок}$ < 3…5-ти років, то фінансування даної наукової розробки є доцільним.

5.5 Висновки

В даному розділі було виконано оцінювання комерційного потенціалу розробки програмного додатку селективного аналізу і синтезу звукових сигналів.

Проведено технологічний аудит з залученням двох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки є високим.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 31509,37 грн. Про абсолютну ефективність вкладених інвестицій в сумі 322252,56 свідчить отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених у наукову розробку інвестицій складе 123%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 70%. Це може свідчити про зацікавленість інвесторів у фінансуванні нової розробки. Термін окупності вкладених у реалізацію проекту інвестицій становить 0,81 року, що також свідчить про доцільність фінансування нової розробки.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було розроблено методи та програмну системи для селективного аналізу та синтезу звукових сигналів.

В результаті виконання кваліфікаційної роботи, було отримано наступні результати:

1. Проведено аналіз предметної галузі, в результаті якого було зроблено висновок про можливість обробки звукових сигналів за допомогою селективного аналізу та синтезу та необхідність розробки власного методу, який буде його реалізовувати.

2. Проведено аналіз існуючих аналогів, який підтвердив актуальність розробки власного програмного додатку, який буде нівелювати недоліки існуючих та реалізовувати розроблені методи.

3. Розроблено метод селективного аналізу звукових сигналів на основі врахування амплітудних спектрів з використанням методів багатопотокового програмування, що дає можливість швидко визначити спектр частот.

4. Розроблено метод синтезу звукових сигналів на основі методу робити зі звуком у браузері Web Audio API, використовуючи при цьому методи реактивного та багатопотокового програмування, що дає можливість підвищити швидкодію роботи веб-додатку та оптимізувати використання технічних засобів.

5. Розроблено структурну схему програмної системи та графічні схеми інтерфейсу. Було розроблено програмний систему за допомогою мови програмування TypeScript, framework Angular, а також бібліотеки Web Audio API та RxJs в середовищі розробки WebStorm.

6. Проведено тестування роботи розробленого програмного додатку, яке підтвердило його стабільність та придатність до використання.

7. Проведено технологічний аудит розроблених методів і засобів селективного аналізу та синтезу звукових сигналів, в ході якого було

встановлено високий технічний рівень та комерційний потенціал розробки. Також, виконано розрахунок економічного ефекту від можливого впровадження розроблених методів і засобів, в ході якого підтверджено доцільність фінансування цієї наукової розробки.

Підсумовуючи усе вищесказане, можна зробити висновок, що задачі магістерської кваліфікаційної роботи було виконано у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Александр Радзишевский. Основы аналогового и цифрового звука: Высшая школа, 2006. – 280с.

2. Порівняльний аналіз javascript фреймворків для розробки мережевого журналу. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki2018/-paper/view/5232

3. Квєтний Р. Н., Богач І. В., Бойко О. Р., Софина О. Ю., Шушура О.М. Комп'ютерне моделювання систем та процесів. Методи обчислень. Частина 2: навчальний посібник за заг. ред. Р.Н. Квєтного. – Вінниця: ВНТУ, 2012. – 230 с.

4. Глинченко А. С. Цифровая обработка: Красноярск: Изд-во КГТУ, 2001. 199с.

5. Дробик, В.В. Кідалов, В.В. Коваль, Б.Я. Костік, В.С. Лазебний, Г.М. Розорінов, Г.О. Сукач. Цифрова обробка аудіо та відеоінформації у мультимедійних системах: Навчальний посібник.О.В. – К.: Наукова думка, 2008. – 144 с.

6. Л. Квурт, С. Хомич. Методичні матеріали до конспекту лекцій з дисципліни "Системи мультимедіа" для студентів базового напрямку 6.0915 "Комп'ютерна інженерія" – Львів: Національний університет "Львівська політехніка", 2009, 120с.

7. Я. І. Дасюк. Функції комплексної змінної. Перетворення Фур'є та Лапласа: Ін-т змісту і методів навчання. – Л., 1999. 271 с.

8. Швидке перетворення Фур'є. URL: https://uk.wikipedia.org/wiki/-Швидке_перетворення_Фур%27є

9. Лінійний фільтр. URL: https://uk.wikipedia.org/wiki/Лінійний_фільтр

10. Web Audio API. URL: https://developer.mozilla.org/ru/docs/Web/API/-Web_Audio_API

11. Reactive Extensions Library for JavaScript. URL: https://rxjs-dev.firebaseapp.com/

12. Node.js. URL: https://uk.wikipedia.org/wiki/Node.js

13. Битва титанов: Angular 2 vs React. Что выбирают в 2017 году? URL: https://blog.ithillel.ua/articles/bitva-titanov-angular-2-vs-react.-chto-vybirayut-v-2017-godu

14. WebStorm. URL: https://uk.wikipedia.org/wiki/WebStorm

15. Angular 2 URL: http://thewebland.net/development/javascript/angular2/-architecture/

16. Калбертсон Р. Быстрое тестирование.: М.: «Вильямс», 2002. — 374 с.

17. Вільна енциклопедія. Інтерфейс користувача. URL: http://uk.wikipedia.org/-wiki/Інтерфейс_користувача

18. Вільна енциклопедія. Software testing. URL: https://en.wikipedia.org/wiki/-Software_testing

19. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. СПб.: Питер, 2004. — 320 с.

20. Boll S.F. Suppression of Acoustic Noise in Seech Using Spectral Subtraction. IEEE Trans. ASSP, Vol.27, No.2, pp 113-120, 1979.

21. Козловський В. О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт. Вінниця: ВНТУ. 2012.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"____ "_____2019 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка програмної системи селективного аналізу та синтезу звукових сигналів» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. О.М. Рейда

"____"_____2019 р.

Виконав:

_____студент гр.1ПІ-18м Л.П. Стахов

"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка програмної системи селективного аналізу та синтезу звукових сигналів». Згідно отриманого завдання кінцевий програмний продукт може використовуватись офіційною організацією.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № _____ від _____.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення ефективності цифрового аналізу та синтезу звукових сигналів шляхом використання реактивного багатопотокового програмування у веб-додатках.

Об’єктом дослідження є процес селективного аналізу та синтезу звукових сигналів у цифровому вигляді.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

- дослідити тенденції розвитку технологій у сфері обробки звукових сигналів;
- проаналізувати сучасні тенденції у створенні веб програм для обробки звукових сигналів онлайн ;
- проаналізувати існуючі засоби для селективного аналізу і синтезу звукових сигналів;
- запропонувати покращення швидкодії селективного аналізу і синтезу звукових сигналів;
- розробити структуру інтерфейсу;
- створити концепцію програми;
- розробити програмний додаток;
- провести тестування додатку.

4. Технічні вимоги

Вихідні дані до роботи: частота дискретизації аудіо-сигналу; вбудована аудіо-карта з частотою дискретизації 192 kHz і разрядністю ЦАП 24 біта; браузер Google Chrome з підтримкою Web Audio API, двухядерний процесор Intel з підтримкою технології Hyper-threading.

Вихідні методи для модифікації: Web Audio API, бібліотека Rxjs, фреймворк Angular, менеджер пакетів npm або yarn.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану питання та постановка задачі	04.09.2019 – 29.09.2019
2	Розробка методу реалізації селективного аналізу та синтезу звукових сигналів	30.09.2019 – 26.10.2019
3	Розробка програмного забезпечення	27.10.2019 – 3.11.2019
4	Тестування програмного додатку	4.11.2019 – 12.11.2019
5	Економічна частина	13.11.2019 –

		17.11.2019

9. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б Лістинг програмної реалізації

```
import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
import { from, Subject } from 'rxjs';
import { bufferCount, filter, map, tap, throttleTime } from 'rxjs/operators';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  @ViewChild('pathElement') pathEl: ElementRef;
  size = 125;
  magnificationRatio = 1;
  magnifiedSize = this.size * this.magnificationRatio;
  sampleRate = 1024;
  stepSize = (this.size * 2) / this.sampleRate;
  audioContext = new AudioContext();
  audioDataStream$ = new Subject();
  path$ = this.audioDataStream$.pipe(
    filter(s => typeof s !== 'undefined'),
    map((value: number) => value * this.magnifiedSize + this.size),
    bufferCount(this.sampleRate),
    map(values =>
      values
        .map((value: number, index) => [index * this.stepSize, value])
        .reduce(
          (acc, point, i, a) =>
            i === 0
```

```
          ? `M ${point[0]},${point[1]}`
          : `${acc} ${this.bezierCommand(point, i, a)}`,
      ''
    )
  )
);

ngOnInit() {
  navigator.getUserMedia(
    { audio: true },
    stream => {
      const source = this.audioContext.createMediaStreamSource(stream);
      const processor = this.audioContext.createScriptProcessor(1024, 1, 1);

      source.connect(processor);
      processor.connect(this.audioContext.destination);

      processor.onaudioprocess = e => {
        e.inputBuffer.getChannelData(0).forEach(dataPoint => {
          this.audioDataStream$.next(dataPoint);
        });
      };
    },
    err => {
      console.log('ERROR', err.message);
    }
  );

  this.path$.subscribe(path => {
    this.pathEl.nativeElement.setAttribute('d', path);
```

```
    });
  }

  line(pointA: number[], pointB: number[]) {
    const lengthX = pointB[0] - pointA[0];
    const lengthY = pointB[1] - pointA[1];
    return {
      length: Math.sqrt(Math.pow(lengthX, 2) + Math.pow(lengthY, 2)),
      angle: Math.atan2(lengthY, lengthX)
    };
  }

  controlPoint(current: number[], previous: any, next: any, reverse: boolean) {
    const p = previous || current;
    const n = next || current;
    const smoothing = 0.2;
    const o = this.line(p, n);
    const angle = o.angle + (reverse ? Math.PI : 0);
    const length = o.length * smoothing;
    const x = current[0] + Math.cos(angle) * length;
    const y = current[1] + Math.sin(angle) * length;
    return [x, y];
  }

  bezierCommand(point: number[], i: number, a: any[]) {
    const [cpsX, cpsY] = this.controlPoint(a[i - 1], a[i - 2], point, false);
    const [cpeX, cpeY] = this.controlPoint(point, a[i - 1], a[i + 1], true);
    return `C ${cpsX},${cpsY} ${cpeX},${cpeY} ${point[0]},${point[1]}`;
  }
}
```

```
var animationRunning = false;
var analysers = new Array;
var CANVAS_WIDTH = 150;
var CANVAS_HEIGHT = 120;

function updateAnalyser(e) {
        var SPACER_WIDTH = 1;
        var BAR_WIDTH = 1;
        var OFFSET = 100;
        var CUTOFF = 23;
        var ctx = e.drawingContext;
        var canvas = ctx.canvas;
        var numBars = Math.round(canvas.width / SPACER_WIDTH);
        var freqByteData = new Uint8Array(e.audioNode.frequencyBinCount);

        e.audioNode.getByteFrequencyData(freqByteData);
        //analyser.getByteTimeDomainData(freqByteData);

        ctx.clearRect(0, 0, canvas.width, canvas.height);
        ctx.fillStyle = '#F6D565';
        ctx.lineCap = 'round';
        var multiplier = e.audioNode.frequencyBinCount / numBars;

        // Draw rectangle for each frequency bin.
        for (var i = 0; i < numBars; ++i) {
        var magnitude = 0;
        var offset = Math.floor( i * multiplier );
        // gotta sum/average the block, or we miss narrow-bandwidth spikes
        for (var j = 0; j< multiplier; j++)
                magnitude += freqByteData[offset + j];
```

```
        magnitude = magnitude / multiplier;
        var magnitude2 = freqByteData[i * multiplier];
        ctx.fillStyle = "hsl( " + Math.round((i*360)/numBars) + ", 100%, 50%)";
        ctx.fillRect(i * SPACER_WIDTH, canvas.height, BAR_WIDTH, -magnitude);
        }
}

function updateAnalysers(time) {
   var rAF = window.requestAnimationFrame ||
            window.webkitRequestAnimationFrame ||
            window.mozRequestAnimationFrame ||
            window.msRequestAnimationFrame;
   rAF( updateAnalysers );

        for (var i = 0; i < analysers.length; i++)
        updateAnalyser(analysers[i]);
}
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { throwError, of } from 'rxjs';
import { map, tap, concatMap, mergeMap, switchMap, shareReplay, catchError }
from 'rxjs/operators';

import { Supplier } from './supplier';

@Injectable({
  providedIn: 'root'
})
```

```
export class SupplierService {
  suppliersUrl = 'api/suppliers';

  // All Suppliers
  suppliers$ = this.http.get<Supplier[]>(this.suppliersUrl)
    .pipe(
      tap(data => console.log('suppliers', JSON.stringify(data))),
      shareReplay(1),
      catchError(this.handleError)
    );

  suppliersWithMap$ = of(1, 5, 8)
    .pipe(
      map(id => this.http.get<Supplier>(`${this.suppliersUrl}/${id}`)
      )
    );

  suppliersWithConcatMap$ = of(1, 5, 8)
    .pipe(
      tap(id => console.log('concatMap source Observable', id)),
      concatMap(id => this.http.get<Supplier>(`${this.suppliersUrl}/${id}`))
    );

  suppliersWithMergeMap$ = of(1, 5, 8)
    .pipe(
      tap(id => console.log('mergeMap source Observable', id)),
      mergeMap(id => this.http.get<Supplier>(`${this.suppliersUrl}/${id}`))
    );
```

```
  suppliersWithSwitchMap$ = of(1, 5, 8)
    .pipe(
      tap(id => console.log('switchMap source Observable', id)),
      switchMap(id => this.http.get<Supplier>(`${this.suppliersUrl}/${id}`))
    );

  constructor(private http: HttpClient) {

  }

  private handleError(err: any) {

    let errorMessage: string;
    if (err.error instanceof ErrorEvent) {

      errorMessage = `An error occurred: ${err.error.message}`;
    } else {

      errorMessage = `Backend returned code ${err.status}: ${err.body.error}`;
    }
    console.error(err);
    return throwError(errorMessage);
  }

}

export interface SynthMessage { }
export class SynthNoteMessage implements SynthMessage {
```

```
  readonly noteNumber: number;
  constructor(noteNumber: number) {
    this.noteNumber = noteNumber;
  }
}
export class SynthNoteOn extends SynthNoteMessage {
  constructor(public velocity: number) { }
}
export class SynthNoteOff extends SynthNoteMessage { }
export class TriggerSample implements SynthMessage {
  public instrument: string;
  public velocity: number;
  constructor(instrument: string, velocity: number) {
   this.instrument = instrument;
   this.velocity = velocity;
  }
}
export class VolumeChange {
  public level: number;
  constructor(level: number) {
    // hack due to arduino stupidity kenny
    this.level = Math.min(level / 127.0);
  }
}
export class WaveformChange {
  public waveForm: string;
  constructor(public rawValue: number) {
    switch (rawValue) {
```

```
      case 0:
        this.waveForm = 'sawtooth';
        break;
      case 1:
        this.waveForm = 'sine';
        break;
      case 2:
        this.waveForm = 'triangle';
        break;
      case 3:
        this.waveForm = 'square';
        break;
      default:
        this.waveForm = 'sawtooth';
    }
  }
}

export class Note {
  ...
  constructor(public noteValues: string[],
              private frequency: number) {
    ...
  }
  noteOn() { .. }
  noteOff() { .. }
}
@Injectable()
```

```
export class SynthesizerService {
  // hold onto our notes once created
  private notes: Note[];
  setup(audioContext: Context, synthStream$: Subject<SynthMessage>,
        audioBus: AudioNode) {
    // start by setting static values for all notes
    Note.configure(audioContext, synthStream$, audioBus);
    // now, configure each note object
    this.notes = [
      new Note(['C0'], 16.3516),
      new Note(['C#0', 'Db0'], 17.3239),
      new Note(['D0'], 18.3540),
      new Note(['D#0', 'Eb0'], 19.4454),
      ...
      new Note(['D8'], 4698.64),
      new Note(['D#8', 'Eb8'], 4978.03),
      new Note(['E8'], 5274.041),
      new Note(['F8'], 5587.652)
    ];
  }
}
const subscription =
   myObservable.subscribe(onMessage, onError, onComplete);
function onMessage(message: MessageType) {
  // do something with the message
}
function onError(error: Error) {
  console.log(error);
```

```
}
function onComplete() {
  // do cleanup if needed
}
// In two seconds, stop listening
setTimeout(() => {
  subscription.unsubscribe();
}, 2000);

function generateNumbers(): Subject<number> {
    // send a value every 1/10th of a second, quit after 20 seconds
    const mySubject: Subject<number> = new Subject();
    let counter = 0;
    const interval = setInterval(() => {
    counter = counter + 1;
    mySubject.next(counter);
    }, 100);
    setTimeout(() => {
        clearInterval(interval);
    }, 20000);
    return mySubject;
}

noteOn() {
    const now = Note.context.currentTime;
    this.createOscillatorAndGainNode();
    this.oscillator.start(0);  // 0 = now
    // ramp up during attack time (how long to take
```

```
    // to get the note to full volume).
    this.gainNode.gain.linearRampToValueAtTime(
        this.volume, now + this.attack);
    // ramp down to sustain volume in future
    this.gainNode.gain.setTargetAtTime(
        this.volume / 2, now + this.attack + this.sustain, 0.5);
}
noteOff() {
    const now = Note.context.currentTime;
    this.gainNode.gain.setTargetAtTime(
        0, now + this.attack + this.sustain + this.decay, 0.5);
}
private createOscillatorAndGainNode(): void {
    this.gainNode = Note.context.createGain();
    this.oscillator = Note.context.createOscillator();
    this.oscillator.type = this.waveform;
    this.oscillator.frequency.value = this.frequency;
    this.oscillator.connect(this.gainNode);
    this.gainNode.connect(Note.audioBus);
    this.gainNode.gain.value = 0;
}
sendNote(note: number | string, duration: number = 300) {
    const self = this;
    this.synthStream$.next(new SynthNoteOn(note));
    setTimeout(() => {
      self.synthStream$.next(new SynthNoteOff(note));
    }, duration);
}
```

```
private processMusicNoteMessage(midiChannelMessage: any) {
   switch (midiChannelMessage.data[0]) {
   case 144:
      this.synthStream$.next(
         new SynthNoteOn(midiChannelMessage.data[1]));
      break;
   case 128:
      this.synthStream$.next(
         new SynthNoteOff(midiChannelMessage.data[1]));
      break;
   }
}
```

Додаток В. Ілюстративний матеріал

Ілюстративний матеріал до захисту магістерської кваліфікаційної роботи

Завідувач кафедри ПЗ, д. т. н., професор _____ О. Н. Романюк

Науковий керівник, к. т. н., доцент кафедри ПЗ _____ О. М. Рейда

Рецензент, к. т. н., доцент кафедри КН _____ О. К. Колесницький

Нормоконтроль, к. т. н., доцент кафедри ПЗ _____ О. М. Рейда

Виконавець, студент гр.1ПІ-18м _____ Л. П. Стахов

2

Мета, предмет та об’єкт дослідження

Метою магістерської кваліфікаційної роботи є підвищення ефективності цифрового аналізу та синтезу звукових сигналів шляхом використання реактивного багатопотокового програмування у веб-додатках.

Об’єкт дослідження: процес селективного аналізу та синтезу звукових сигналів у цифровому вигляді.

Предмет дослідження: методи та засоби аналізу та синтезу звукових сигналів.

Слайд 2 – Мета, предмет та об’єкт дослідження

3

Актуальність теми

Широке використання звукових сигналів в інтернет просторі

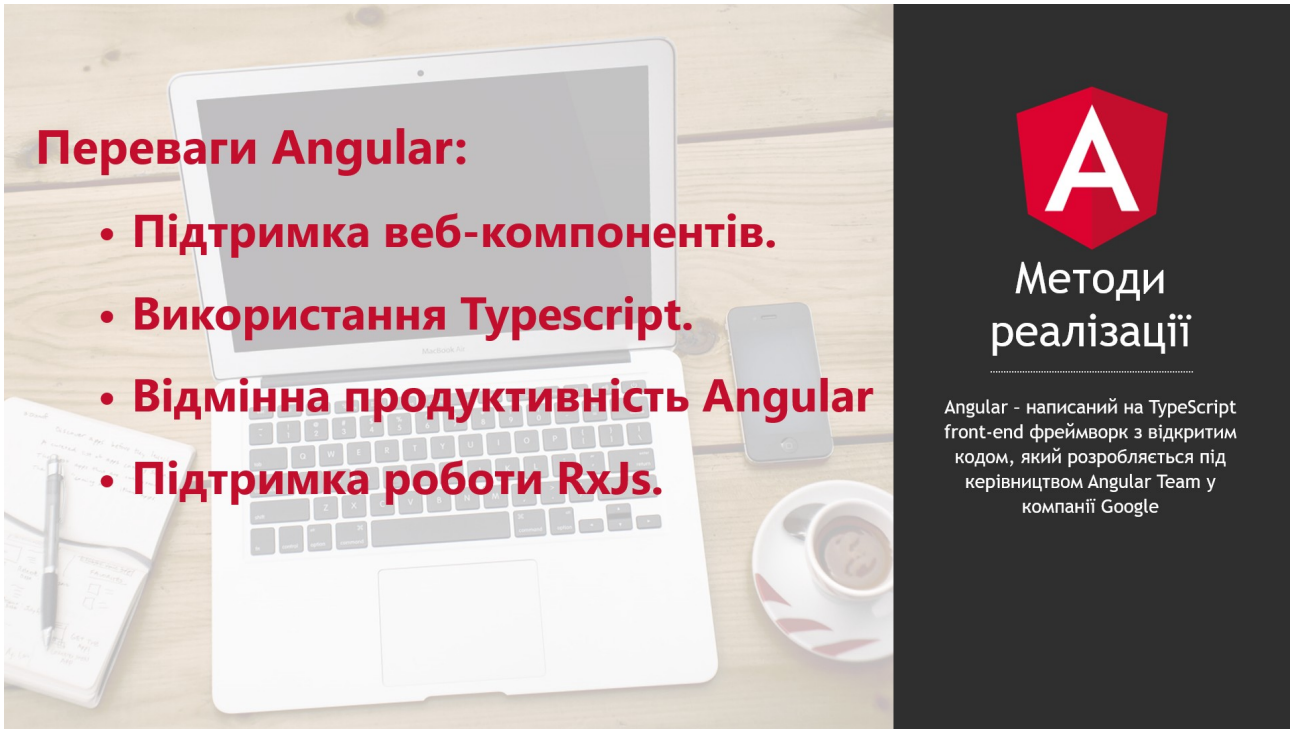
Слайд 3 – Актуальність розробки

Наукова новизна розробки

1 Вперше запропоновано багатопотоковий метод селективного аналізу звукових сигналів у веб-додатках, особливість якого полягає у використанні паралельних багатопроцесорних методів

2 Вперше запропоновано комбінований метод виконання синтезу звукових сигналів із використанням технологій реактивного та багатопотокового програмування Rxjs

Слайд 4 – Наукова новизна розробки



Слайд 5 – Методи реалізації

6

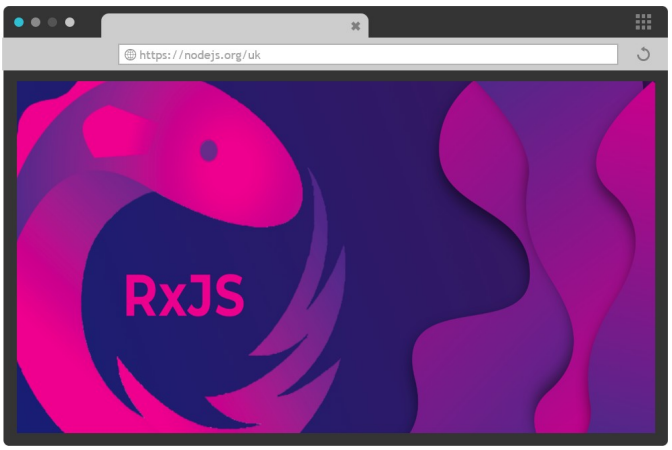
Методи реалізації

Web Audio API

Web audio API - потужний інструмент для маніпуляції звукової складової на веб-сторінці, що дає можливість розробникам вибрати джерела, можливість отримувати частоту, форму хвилі і інші дані з звукового джерела, додати до них спеціальні звукові ефекти

RxJs

RxJS - це бібліотека для JS, яка використовує патерн Observable (з англ. "Оглядач") для спрощення обробки і компоновки асинхронного або callback коду



Слайд 6 – Методи реалізації

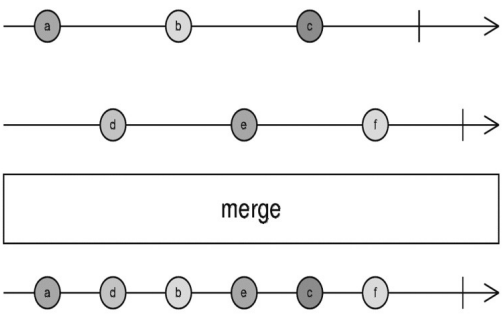
7

Переваги методів реалізації

Переваги:

- Покращена реакція програми
- Більш ефективне використанням багатопроцесування
- Покращена структура програми
- Ефективне використання ресурсів системи

Поєднавши вже існуючі реалізації обробки цифрових сигналів з реактивним багатопотоковим програмуванням, можна досягти підвищення швидкодії і оптимізації ефективності використання технічних засобів



Слайд 7 – Переваги методів реалізації

8

Розробка методу селективного аналізу

Алгоритм вирахування амплітудних спектрів

Спектральна щільність:

$$Pss\ (iw) = Pxx\ (iw) - Pnn\ (iw) \qquad (2.2)$$

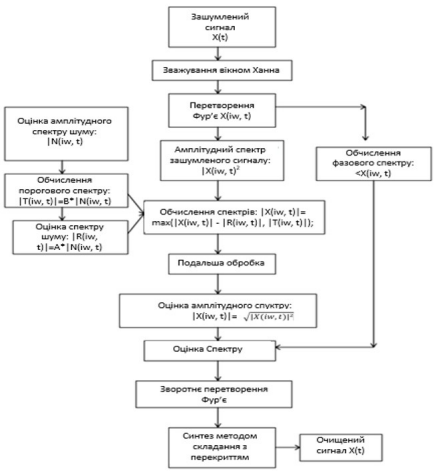
Співвідношенням спектрального віднімання:

$$|S(t,iw)|^2 = \begin{cases} |X_i(t,iw)|^2, \text{якщо } |X_i(t,iw)|^2 \geq (A(t) + B|N(t,iw)|^2) \\ B|N(t,iw)|^2, \quad \text{інакше} \end{cases} \qquad (2.3)$$

Сценарій використання вирахування спектрів складається з трьох основних частин:

1. Вибору параметрів обробки (за потреби),
2. Виконання навчання на спектр шуму
3. Виконання процедури вирахування амплітудних спектрів

Блок-схема алгоритму вирахування амплітудних спектрів



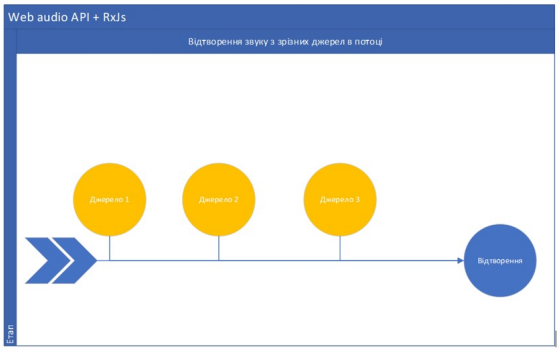
Слайд 8 – Розробка методу селективного аналізу

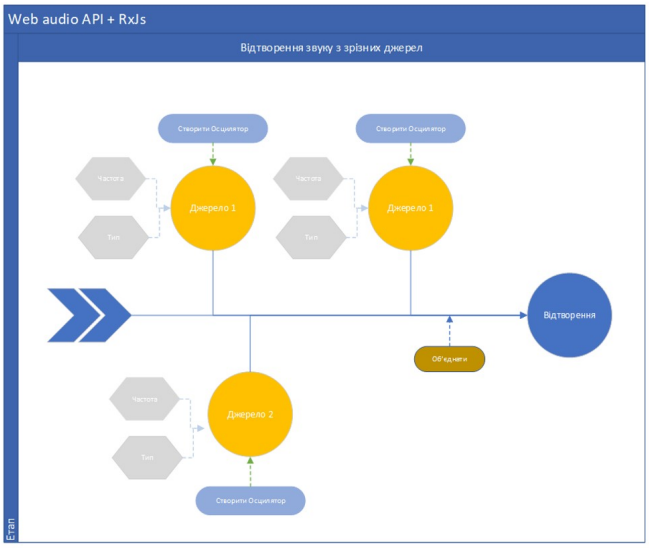
9

Розробка методу синтезу звукових сигналів

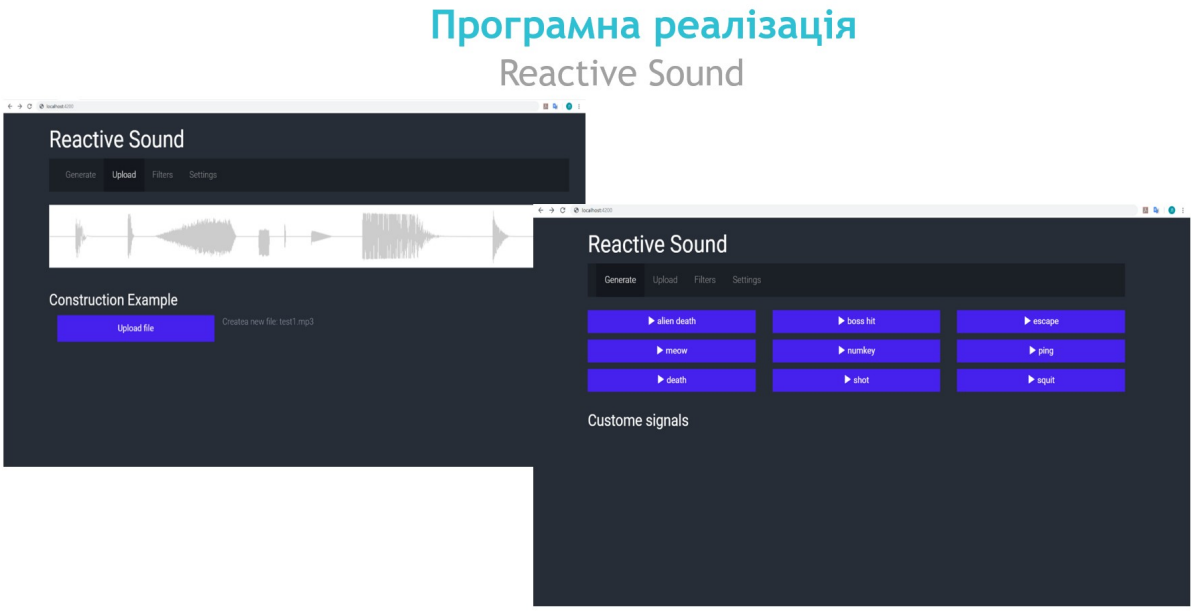
Генерація, обробка і відтворення

Використання реактивного потокового методу створення та відтворення звукових сигналів підвищує швидкодію та оптимізує використання технічних засобів

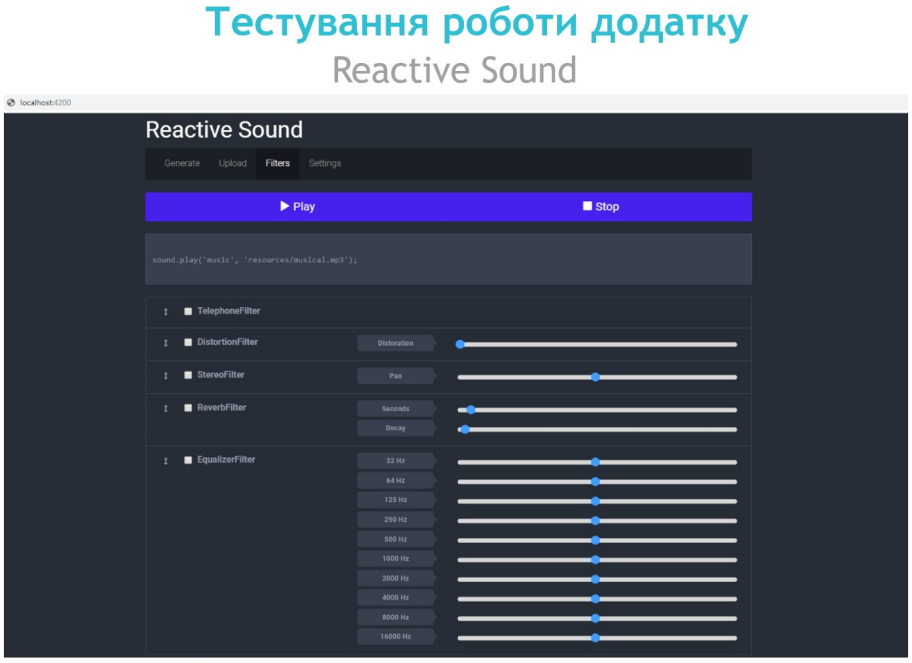




Слайд 9 – Розробка методу чинтезу звукових сигналів



Слайд 10 – Інтерфейс програмної реалізації



Слайд 11 – Тестування роботи додатку

Результати роботи та висновки

12

- ✓ Проведено аналіз предметної галузі.
- ✓ Проведено аналіз існуючих аналогів, який підтвердив актуальність розробки.
- ✓ Розроблено метод селективного аналізу звукових сигналів на основі врахування амплітудних спектрів з використанням методів багатопотокового програмування.
- ✓ Розроблено метод синтезу звукових сигналів;
- ✓ Розроблено структурну схему програмної системи та графічні схеми інтерфейсу.
- ✓ Проведено тестування роботи розробленого програмного додатку.
- ✓ Проведено технологічний аудит розроблених методів і засобів селективного аналізу та синтезу звукових сигналів.

Слайд 12 – Результати роботи та висновки