НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.02

«ПОГОДЖЕНО»
Декан факультету
інформаційних технологій
Глазунова О.Г., д.п.н., професор
_____
____ _____ 2023 р.

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних наук
Голуб Б.Л., к.т.н., доцент
_____
____ _____ 2023 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Методи обробки звукового сигналу при використанні гітари як контролера в веб-орієнтованих застосунках

Спеціальність Інженерія програмного забезпечення

(код і назва)

Освітня програма Програмне забезпечення інформаційних систем

(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____ _____ _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

кандидат ф.-м. наук, доцент _____ Кириченко В.В
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Недьошев М.В
(підпис) (ПІБ студента)

КИЇВ-2023

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) _____

ЗАТВЕРДЖУЮ
Завідувач кафедри_____
_____
_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Недьошев Максим Владиславович
(прізвище, ім'я, по батькові)

Спеціальність 121 Інженерія програмного забезпечення
(код і назва)

Освітня програма Інформаційні управляючі системи та технології
(назва)

Орієнтація освітньої програми_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Методи обробки звукового сигналу при використанні гітари як контролера в веб-орієнтованих застосунках

затверджена наказом ректора НУБіП України від "____"_____20___р. №_____

Термін подання завершеної роботи на кафедру "5" листопада 2023
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи _____

Перелік питань, що підлягають дослідженню:

1. _____
2. _____
3. _____

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ Кириченко В.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Недьошев М.В
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Вебзастосунок (або вебдодаток) — розподілений застосунок, в якому клієнтом виступає браузер.

HTML — Hypertext Markup Language — це код, який використовується для структурування вебсторінки та її вмісту.

API — Application Programming Interface — це спосіб для двох чи більше комп’ютерних програм спілкуватися одна з одною.

Нота — звук гітари певної частоти, який має свою назву.

Гітарний ефект — це фізичний прилад чи комп’ютерний алгоритм, який видозмінює звучання гітари.

Disturtion — гітарний ефект, який змінює звучання гітари шляхом «жорсткого» обрізання амплітуди аудіосигналу.

MIDI — стандарт передачі інформації між електронними музичними інструментами.

ВСТУП

Сьогодні важко уявити сферу бізнесу, яка б не користувалася вебтехнологіями для свого розвитку. Веборієнтовані застосунки стали важливою частиною нашого життя, перетворюючи багато аспектів бізнесу, включаючи музичну освіту, в більш інтерактивні та цікаві.

Сучасний ріст технологій та поширення веборієнтованих застосунків вимагає від розробників виходу за межі звичних інтерфейсів. Так зараз інтегруються тривимірна графіка, технології доповненої реальності, нейронні мережі та інше. Однак, музична освіта використовує малий відсоток доступних технологій у веббраузері.

Гітара є одним з найпопулярніших музичних інструментів, і вивчення її гри є складним завданням, як і навчання інших мов. Під час вивчення музики важливі як практичні, так і теоретичні навички, такі як вивчення нот, розуміння патернів, відчуття ритму та вміння грати на інструменті. Існує проблема відсутності інтерактивності у музичних онлайн-курсах.

Мета цього дослідження — дослідити можливість використання гітари як контролера для веборієнтованих застосунків та розробити систему, яка забезпечить більш інтуїтивну та ефективну взаємодію користувачів з вебзастосунками. Для досягнення такої мети було поставлено наступні завдання:

- Створити метод для аналізу звуку з аудіо джерела вводу при прямому підключенні гітари до пристрою з веббраузером;
- Проаналізувати особливості роботи користувацьких інтерфейсів в веборієнтованих застосунках з використанням гітари як контролера;

- Розробку архітектури системи взаємодії гітари з веборієнтованими додатками;
- Створити бібліотеку для використання гітари як контролера;
- Побудувати демонстраційний додаток для навчання грі на гітарі з перевіркою коректності виконаної ноти та з вбудованими гітарними ефектами.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд способів взаємодії з комп’ютером за допомогою гітари

Сьогодні безліч сфер перейшли цифровізацію: від літератури, кіно і освіти до банкінгу. Звісно індустрія музики є однією з перших хто пройшов цифровізацію. Так, перший музичний запис був виконаний у 1951 році, задовго до появи масового персонального комп’ютера. А створення музики завдяки програмам почало свій шлях з 1957-го року. У 1990-х роках комп’ютери почали використовуватися разом з музичними інструментами, почали впроваджуватися так звані “плагіни”, які мали змінювати звук музичного інструменту під час запису.

Сьогодні музикальні застосунки для взаємодії з гітарою можна поділити на такі категорії:

- Застосунки для генерації звуків (наприклад, Ample Sound Ample Guitar M Lite II);
- Застосунки для налаштування музичних інструментів, або, коротко кажучи “тюнери” (наприклад, GuitarTuna);
- Застосунки з ефектами (наприклад, GuitarRig);
- Застосунки для вивчення композицій (наприклад, Songsterr);
- Застосунки для міксу та редагування музичних композицій (наприклад, Apple GarageBand);
- Комп’ютерні ігри (наприклад, Rocksmith);

Більшість таких застосунків є застосунками для персональних комп’ютерів, які потрібно встановлювати.

Як контролер гітару найкраще використовує програма Rocksmit, розроблена у 2007 році для персонального комп’ютера. Програма йде з власним кабелем для підключення інструменту. В програмі виконується аналіз звуку для перевірки коректності ноти, а інтерфейс нагадує програму GuitarPro, яка має взагалі спеціальний контролер, який не відтворює звук. В таких програмах потрібно вчасно натискати потрібні кнопки. Такий жанр ігор називається ритм-ігри. Вони не завжди потребують спеціального обладнання (наприклад, osu!), але потребують від користувача гарних рефлексів та почуття ритму. Для гітари, такий жанр ігор є досить корисним, оскільки дає змогу здобути не тільки практичні навички, а також і теоретичні при правильному підході.

1.2. Огляд сучасних методів управління вебзастосунками

Розробка програмного забезпечення в сучасному світі нерозривно пов'язана зі стрімким розвитком технологій та ростом вимог користувачів. Взаємодія з програмами стає все більш складною та різноманітною, що вимагає розробників знаходити нові та цікаві способи взаємодії. Спочатку коли вебсторінки являли собою просту HTML сторінку без інтерактивності у комп’ютерному світі домінували настільні додатки, які потрібно було встановлювати. Такі додатки мають доступ майже до усіх ресурсів персонального комп’ютера. З ростом популярності інтернету вебсайти ставали складнішими та більш інтерактивними. Так до вебсайтів переїхали звичні додатки з персональних комп’ютерів, наприклад, текстові (Google Docs) та графічні редактори (Online Photoshop), месенджери (Microsoft Teams) та навіть середовища розробки (Visual Studio Code) та тривимірні редактори (Spline). Дуже стрімко розвивають VR та AR технології, що покращать інтерактивний досвід використання вебдодатків. У цьому контексті зростає інтерес до використання

нетрадиційних контролерів, які можуть забезпечити більш натуральну, інтуїтивну та ефективну взаємодію.

З ростом складності інтерфейсів підвищується складність створювання цих інтерфейсів. У сучасній веброзробці створюється безліч способів підвищення розробки: додавання нових функцій та інструментів до мови програмування JavaScipt, побудова інструментів над мовою (TypeScript), розробка бібліотек (Lodash, AnimeJS, ThreeJS, d3.js) та повноцінні фреймворки (React, Vue, Astro) - такі інструменти являються стандартом та важко уявити сучасний продукт без таких інструментів. Стосовно роботи з аудіо ведуться робота на WebAudio - браузерного API для роботи зі звуком. Щодо бібліотек є tone.js, howler.js.

Використання музичних інструментів у ролі контролерів в програмному забезпеченні не є новим явищем. Такий підхід дозволяє не лише розширити можливості використання інструментів, а й надати користувачам новий рівень взаємодії з технологіями.

1.3. Аналіз можливостей гітари як потенційного контролера.

Гітара – це один з найпопулярніших музичних інструментів, який використовується в різних музичних жанрах та культурах. Вона компактна, проста у виготовленні та покриває більшість потрібного музичного діапазону, завдяки чому стала улюбленою обробкою для багатьох музикантів.

За останні двадцять роки вже було зроблено певні кроки у напрямку використання гітари як контролера переважно у галузі розваг. Наприклад, у музичних застосунках для персональних комп’ютерів гітара може використовуватися як інтерфейс для відтворення музики та контролю звукових

ефектів. Однак використання гітари в якості контролера для веборієнтованих додатків ще не було належним чином досліджене та впроваджене. Загалом є онлайн тюнери.

Розробка технології використання гітари як контролера для веборієнтованих застосунків потребує вирішення певних технічних, дизайнерських та взаємодійних проблем. Стоїть завдання створити ефективний та зручний інтерфейс, який дозволить користувачам взаємодіяти з вебдодатками за допомогою гітари. Такий інтерфейс повинен бути гнучким, інтуїтивно зрозумілим та забезпечувати високий рівень відгуку на дії користувача.

У гітари є широкий спектр відтворення звуків, які можуть бути використані як команди для взаємодії з програмними продуктами. Наприклад, рухи струнами або ритмічні патерни можуть слугувати командами для виконання різних дій, від відтворення звуку до навігації по інтерфейсу.

Використання гітари як контролера вимагає вирішення ряду технічних проблем, точніше розробка алгоритмів обробки даних та їх інтерпретація.

Використання гітари як контролера надає природну взаємодію з користувачем, можливість використання музичних звуків для команд, а також потенціал для створення вебзастосунків, які виділяються.

1.4. Визначення плюсів і мінусів використання гітари у веборієнтованих застосунках

Використання гітари як контролера у веборієнтованих застосунках має свої переваги та недоліки.

Плюси використання гітари у веборієнтованих застосунках:

- Унікальність інтерфейсу: Гітара є унікальним інтерфейсом, який може надати новий рівень взаємодії користувача з вебзастосунками, що може збільшити зацікавленість користувача продуктом;
- Можливість реалізації нових ідей: Використання гітари як контролера відкриває двері для реалізації нових креативних ідеї та концепцій, наприклад, нового способу навчання чи сумісної практики;

Мінуси використання гітари у веборієнтованих застосунках:

- Складність взаємодії: Використання гітари як контролера може бути складним для користувачів, які не володіють навичками гри на гітарі. Швидкість вводу інформації таким спобом є дуже маленькою.
- Неуніверсальність: Гітара звичайно є неуніверсальним контролером і не підходить для всіх типів вебдодатків.
- Обмежені можливості в порівнянні з іншими контролерами: Гітара може бути обмеженою у функціональності порівняно з іншими сучасними контролерами, такими як сенсорні панелі чи звичайна клавіатура.
- Потреба в додатковому обладнанні: Для використання гітари як контролера, користувачам може знадобитися додаткове обладнання.

Аналіз плюсів та мінусів використання гітари у веборієнтованих застосунках допомагає визначити, наскільки цей підхід може бути ефективним для конкретного застосунку та спрямовує подальший процес дослідження та розробки.

1.5. Формулювання завдань дослідження

Для ефективної імплементації використання гітари як контролера у веборієнтованих застосунках, необхідно ретельно спроектувати модель

взаємодії. Функціональний підхід зосереджується на визначенні основних функцій та операцій, які гітара може виконувати в системі. Для цього можна використовувати діаграми прецедентів та послідовності, що допоможе зрозуміти, як користувач буде взаємодіяти з додатком через гітару.

Важливо визначити, яким чином буде взаємодіяти програмний код додатка з даними, що надходять від гітари. Це включає розробку алгоритмів обробки даних, виявлення звуків, а також визначення способів передачі даних між гітарою та комп'ютером.

Важливою частиною моделювання є проєктування інтерфейсу, який дозволить користувачам комфортно та ефективно взаємодіяти з додатком через гітару. Дизайн інтерфейсу повинен бути інтуїтивно зрозумілим, а також забезпечувати можливість налаштування параметрів взаємодії під потреби користувачів.

Розробка системи містить процес проєктування архітектури, яка буде підтримувати взаємодію гітари як контролера з веборієнтованими додатками. Систему можна поділити на різні компоненти, які відповідають за різні аспекти взаємодії гітари з додатком. Це можуть бути компоненти для зчитування даних з гітари, відтворення звуку, інтерфейсу користувача тощо. Важливо ретельно розглянути структуру цих компонентів та їхню взаємодію.

Оскільки гітара може генерувати різні сигнали при грі, важливо розробити алгоритми обробки цих даних для точного аналізу звуку. Алгоритми повинні бути ефективними та надійними, щоб забезпечити плавну та точну взаємодію з додатком.

Система повинна бути здатною інтегруватися з різними веборієнтованими додатками, такими як ігри, музичні програми, віртуальні інструменти тощо. Це

включає розробку інтерфейсів програмування додатків (API), які дозволять взаємодіяти з системою зовнішнім програмам.

2. МОДЕЛЮВАННЯ СИСТЕМИ

2.1. Поверхневе моделювання системи

Перед початком моделювання системи потрібно визначитися з основними вимогами та цілями дослідження. Основне завдання роботи це розробити програмний інструмент який можна використовувати для побудови веборієнтованих додатків, які використовують гітару як інструмент вводу інформації. Перед тим як розробляти інструмент для роботи з гітарою у веборієнтованих застосунках потрібно виявити як саме цей інструмент буде використовуватись.

З точки зору бізнесу сучасні вебдодатки звісно орієнтовані на користувача і розв'язувати його проблеми. Найбільш ймовірне використання користувачем гітари це практика власних навичок чи навчання. Іншим варіантом є використання на концерті чи студійний запис музики — для таких завдань краще підійде професійне обладнання, яке видає звук вищої якості ніж комп’ютерні алгоритми. Тому використання гітари у веборієнтованих застосунках доцільно оцінювати в рамках користувача, який не є професіоналом. Такому користувачеві потрібен простий спосіб створювати музику та отримувати навички використання інструменту. Способи використання гітари у веборієнтованому додатку представлено у вигляді діаграми прецедентів на рис. 2.1. З діаграми видно що основними акторами системи є користувач та саме інструмент для роботи з гітарою. Очевидним прецедентом такої програми буде це прослуховування звуку власного виконання на гітарі. В теорії може бути безліч прецедентів використання користувачем цього інструменту, але найбільш очевидні це виконання музичних композицій переглядаючи табулатури — спеціальний спосіб запису мелодії, як ноти, але оптимізоване для гітари. Гітарний контролер у свою чергу призначений для аналізу звуку користувача, а дані після аналізу використовуються вебзастосунком.

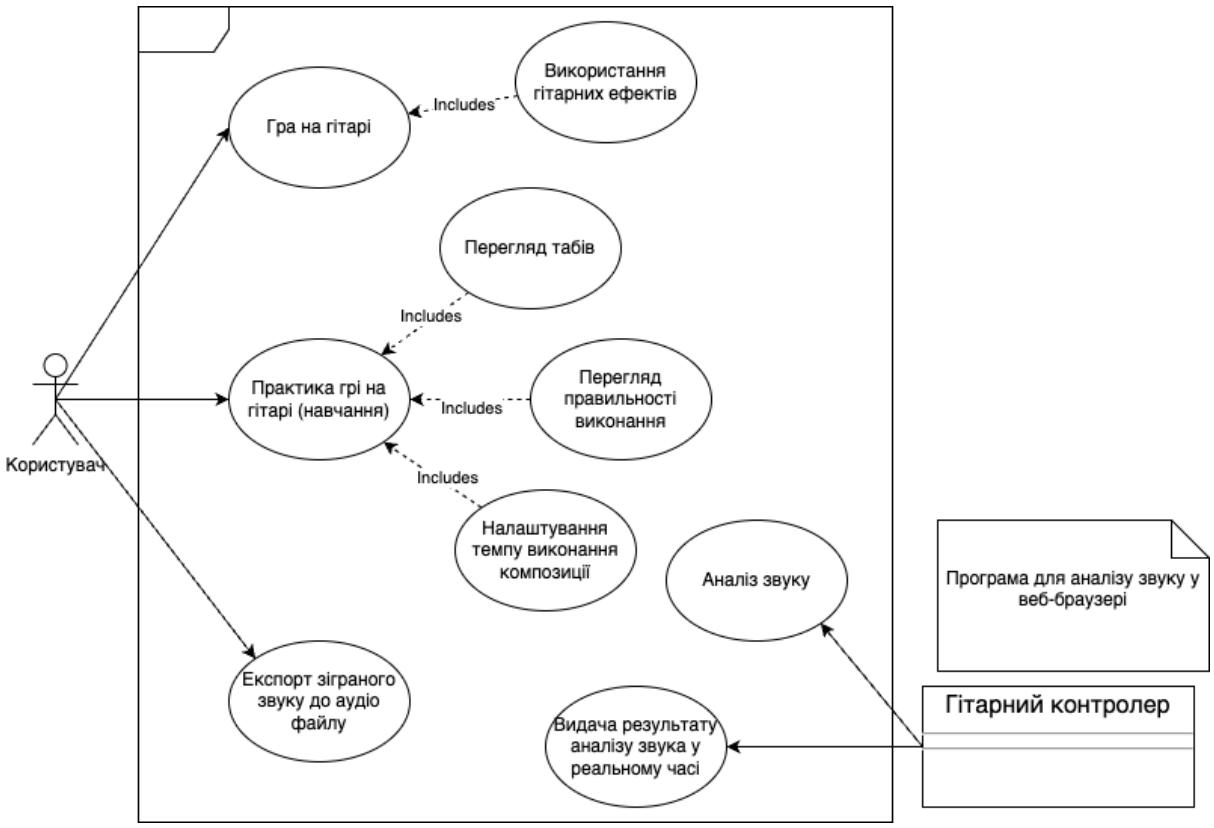

Рисунок 2.1 - діаграма прецедентів додатка для навчання грі на гітарі

Отже, виходячи з цієї діаграми прецедентів можна зробити висновки щодо структури майбутньої системи. У системі повинно бути явно відокремлено компонент, який займається обробкою звуку, завдяки чому можна створювати будь-які інтерфейси з якими користувач зможе взаємодіяти за допомогою гітари. Якщо поглянути на компонент який працює безпосередньо зі звуком, то можна виділи такі прецеденти:

- Обробка звукового сигналу та переведення звуку у масив даних;
- Обробка масиву даних для пошуку зіграної ноти та її гучності;
- Відтворення звуку зі звуковими ефектами.

Ці прецеденти представлено на рис. 2.2.

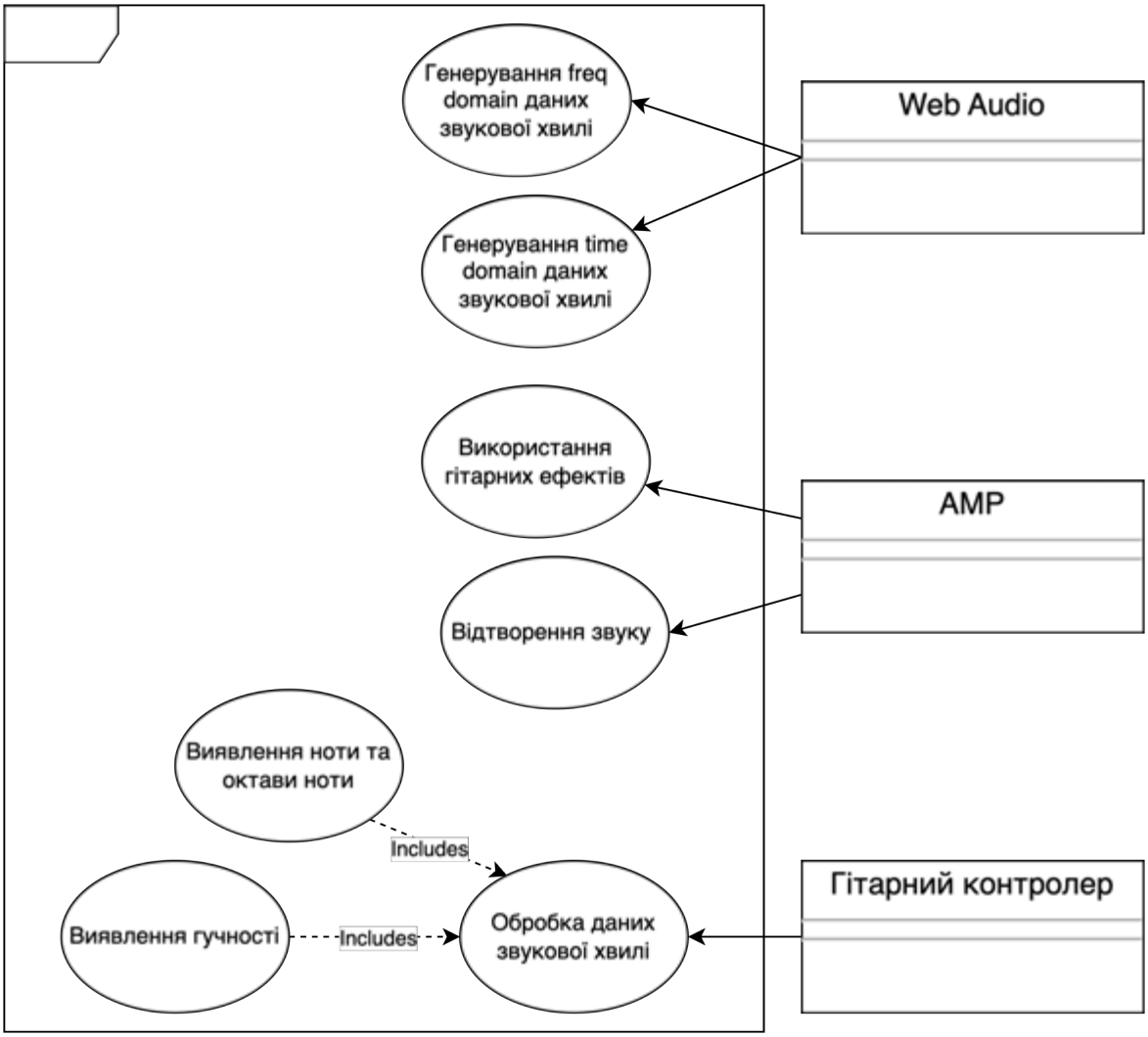

Рисунок 2.2 - Діаграма прецедентів гітарного

Детально взаємодію компонентів показано рис 2.3. у вигляді діаграми “послідовності”. Користувачеві потрібно надати доступ веббраузера до свого мікрофона. Після надання доступу до мікрофона, браузер надає можливість програмістам отримувати дані звукової хвилі у проміжку часу. Потім ці дані аналізуються модулем аналізатором звуку, який знаходить виконану ноту, її

октаву та гучність. Потім ці дані повертаються користувачеві у зручному для нього візуальному інтерфейсі.

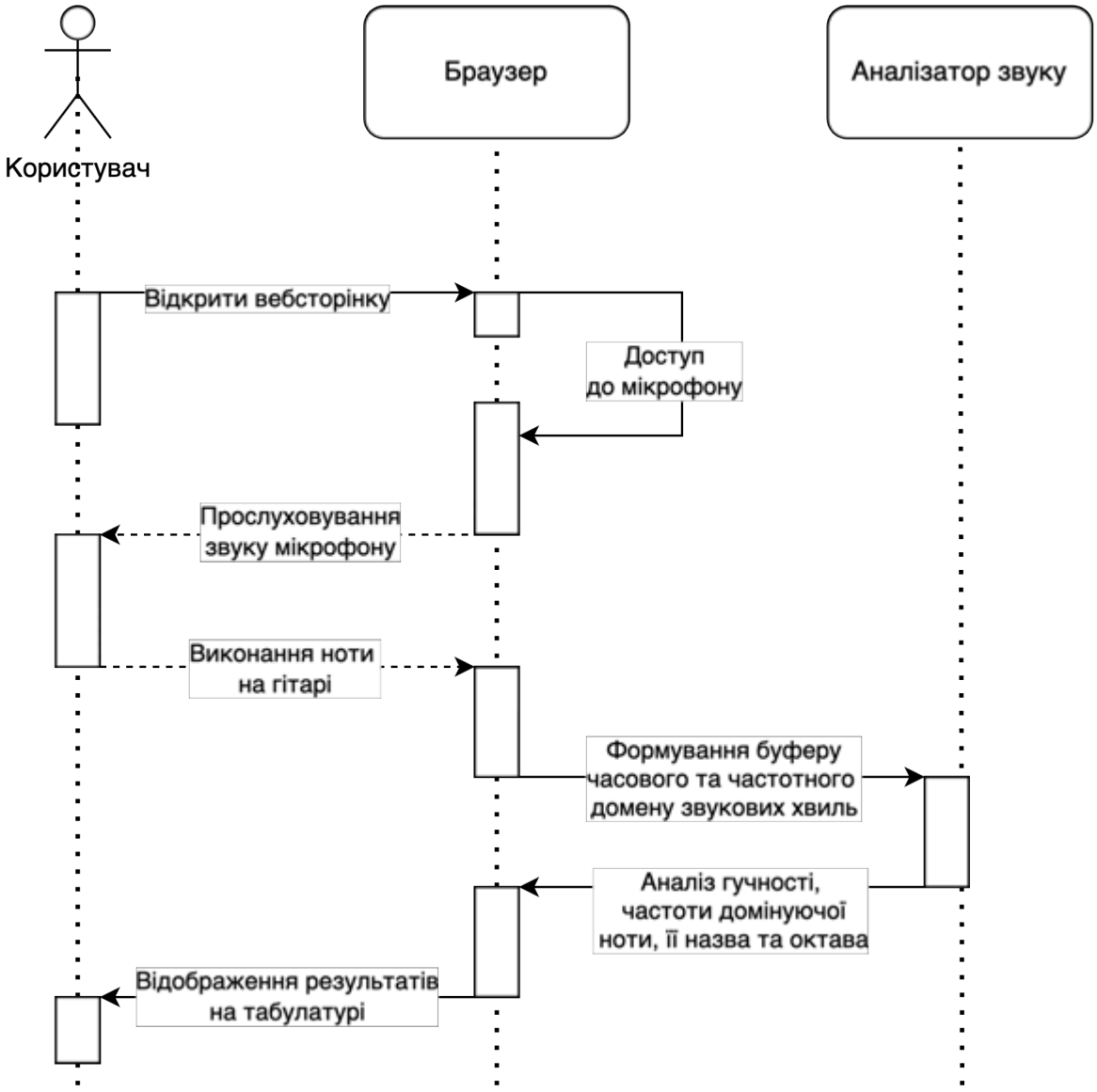

Рисунок 2.3. - діаграма послідовності процесу

Детальніше послідовність дій, які виконуються в рамках кожного прецеденту представлено на діаграмі активності. Отже, дослідивши можливе використання додатка користувачем можна виділити такі компоненти системи:

- Інтерфейс (UI) - вебзастосунок, в якому користувач може взаємодіяти з нотами та отримувати інформацію аналізу виконання звуків на гітарі;
- AMP - компонент, який відтворює звук, використовуючи різні гітарні ефекти;
- PlaySession analyzer - компонент, який призваний аналізувати звук. Саме він використовує гітару як контролер у веборієнтованому застосунку. Таким чином використовуючи гітару можна взаємодіяти з інтерфейсом вебзастосунку (як графічним, так і аудіоінтерфейсом).

Компоненти системи представлено на рисунку 2.4.

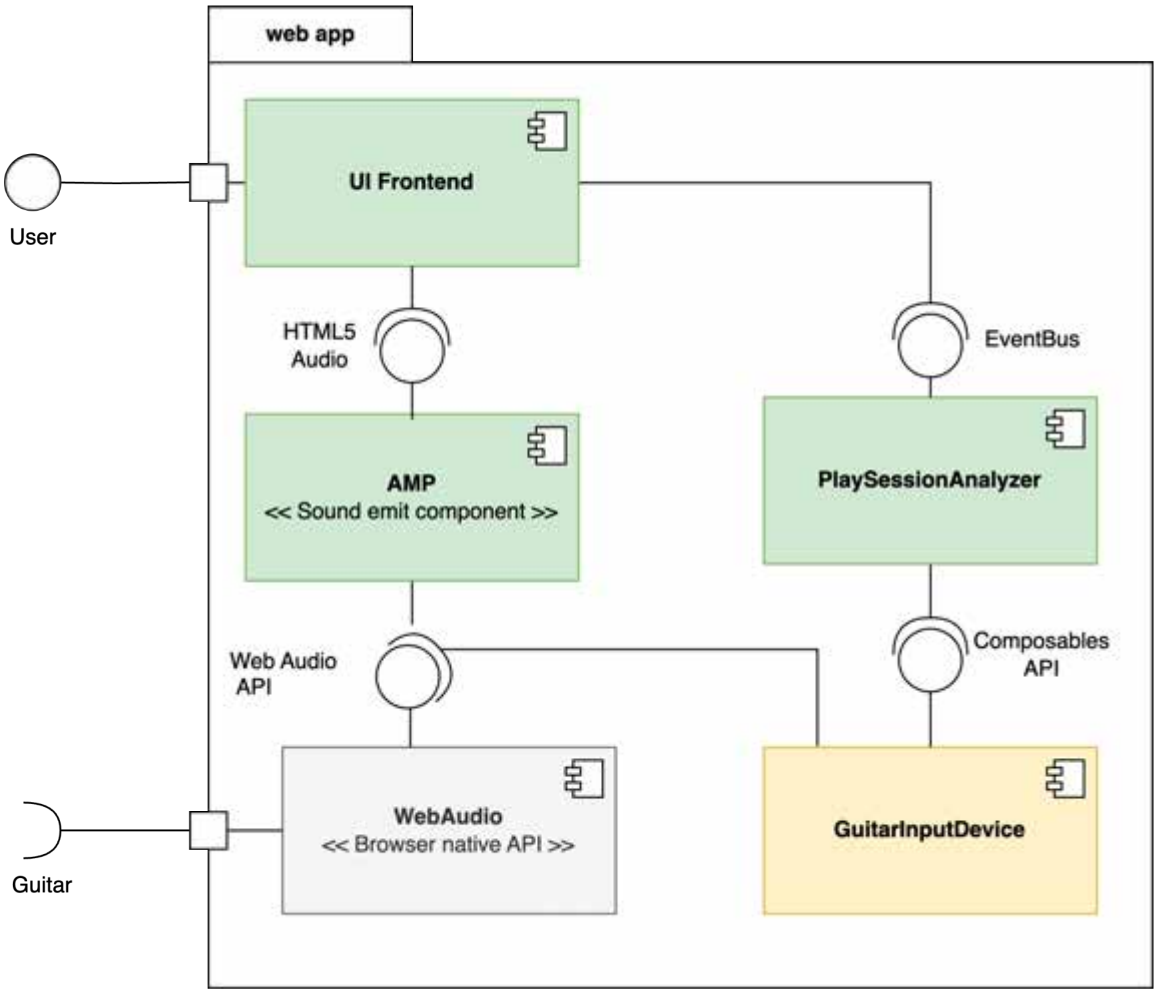

Рисунок 2.4 - діаграма компонентів

Основним компонентом що досліджується є компонент аналізу звуку, оскільки саме використовуючи звук користувач буде взаємодіяти з інтерфейсом веборієнтованих додатків. Такий компонент представлений у вигляді бібліотеки на мові JavaScript - оскільки саме на цій мові можна взаємодіяти з аудіо на вебсторінках. Бібліотека використовує браузерний інтерфейс для взаємодії з мікрофоном для отримання звуку від користувача - WebAudio. Бібліотека повинна містити інструменти для відтворення, модифікування та аналізу гітарного звуку.

Діаграму прецедентів бібліотеки представлено на рисунку 2.5. Основні функції для користувача бібліотеки це:

- Виявлення назви ноти, її октави, частоти та гучності з сигналу мікрофона;
- Додавання гітарних ефектів під час відтворення звуку у веббраузері.

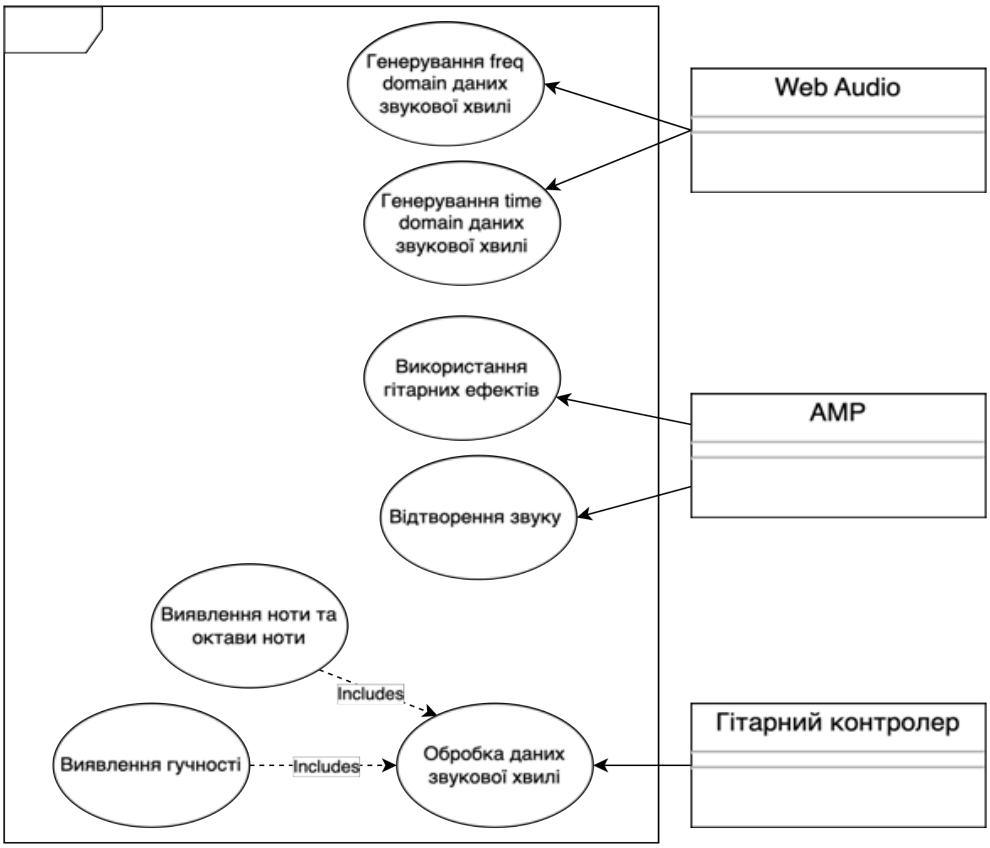

Рисунок 2.5. - діаграма прецедентів гітарного контролера.

Взаємодію користувача з бібліотекою можна представити у вигляді діаграми активності, яка зображена на рисунку 2.6.

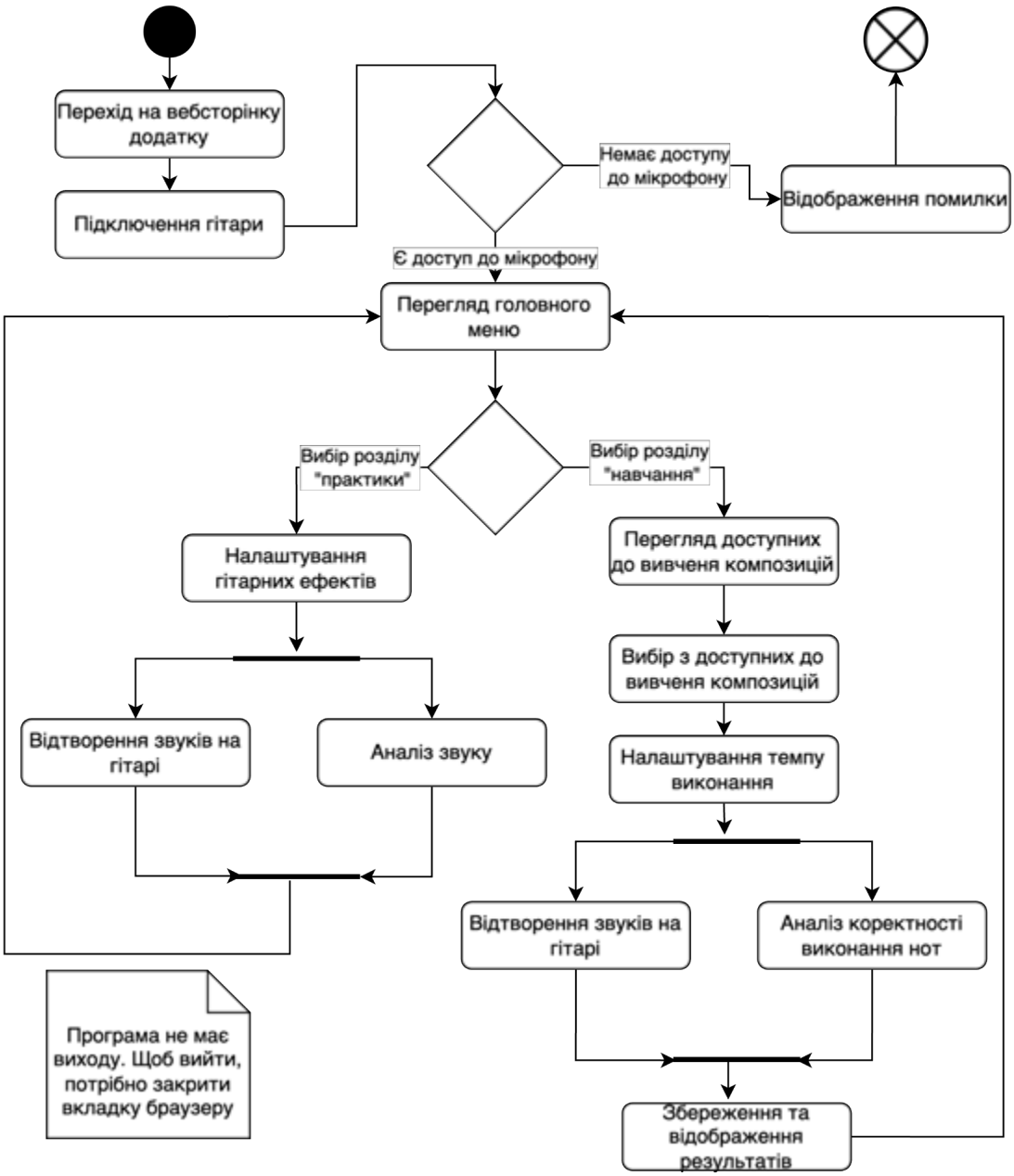

Рисунок 2.6. - діаграма активності роботи веборієнтованого додатка з використанням аналізу гітарного звуку

2.2 Дослідження способів аналізу звукових сигналів

Можна виділити два способи аналізу звуку:

- Методи, засновані на характеристиках спектра звуку — Ці методи використовують спектральний аналіз для визначення ключових характеристик звуку. Вони можуть допомогти в ідентифікації звуків на основі їх спектральних характеристик.

- Штучні нейронні мережі - це комп'ютерні системи, натхненні структурою та функціонуванням мозку. Вони здатні навчатися розпізнавати звукові образи, використовуючи великі набори даних для тренування.

2.2.1. Методи, засновані на характеристиках спектра звуку

Методи, засновані на характеристиках спектра звуку, використовують принципи спектрального аналізу для ідентифікації та розпізнавання різних звукових образів. Ці методи вважаються особливо потужними, оскільки вони здатні видобути ключову інформацію з звукових сигналів, що дозволяє виконувати складне розпізнавання образів.

Однією з основних концепцій у цих методах є використання спектрограми - візуалізації, яка показує, як спектральний вміст сигналу змінюється з часом. Ця техніка дозволяє відобразити зміни в інтенсивності та частоті звуку протягом часу, що може надати значну інформацію про джерело звуку.

Основними аспектами спектрального аналізу є визначення таких характеристик звуку, як висота тона, гучність та тембр. Ці характеристики відіграють ключову роль у розпізнаванні звукових образів:

- Висота тону: Вона визначається основною частотою звукового сигналу і часто використовується для розпізнавання музичних нот та голосових команд.
- Гучність: Ця характеристика, що відображає енергію звукового сигналу, може допомогти в розпізнаванні різних рівнів гучності та їх вплив на звуковий образ.
- Тембр: Він відображає унікальні особливості звукового сигналу, які дозволяють нам відрізнити одне джерело звуку від іншого, наприклад, розпізнати різницю між голосами людей або між різними музичними інструментами.

Такий підхід ідеально підходить для обробки звуку у реальному часі, хоча й звісно проаналізувати виходить лише кілька параметрів.

2.2.2. Методи, засновані на нейронних мережах

Штучні нейронні мережі можна використовувати для аналізу звуку в декількох різних сценаріях, включаючи класифікацію звуків, розпізнавання мови та музичний аналіз. Цей процес зазвичай містить декілька ключових етапів.

1. Підготовка даних: Перед тим, як можна використовувати нейронну мережу для аналізу звуку, потрібно підготувати вхідні дані. Це може включати конвертацію аудіофайлів в придатний для обробки формат, а також застосування передопрацювання, такого як нормалізація та видалення шуму.

2. Екстракція ознак: Після того, як дані підготовлені, можна провести екстракцію ознак. Важливі аудіо характеристики, такі як спектральні ознаки, темпові ознаки або MFCC (Mel Frequency Cepstral Coefficients) - особливості, які відображають форму звукового спектра, можуть бути витягнуті і використані як вхідні дані для нейронної мережі.

3. Навчання мережі: Навчання нейронної мережі включає використання великого набору тренувальних даних, щоб вона могла "вивчити" відповідні характеристики та взаємозв'язки між ними. Цей процес зазвичай містить подачу вхідних даних (ознак) до мережі та коригування ваг мережі за допомогою процесу, відомого як зворотне розповсюдження помилки, щоб мінімізувати різницю між прогнозованим та реальним результатом.

4. Тестування та валідація: Після тренування нейронної мережі її потрібно протестувати на нових даних, щоб перевірити, наскільки добре вона може загально застосовуватися. Це може включати перехресну валідацію та інші техніки для оцінки точності, повноти та інших метрик.

5. Застосування моделі: Після того, як модель була навчена та перевірена, її можна використовувати для розпізнавання або класифікації нових звукових даних в реальному часі.

Особливо важливе застосування нейронних мереж для аналізу звуку — це розпізнавання мови, де мережі можуть бути навчені розпізнавати та перетворювати звукові сигнали на текст. Інший важливий напрямок — це музичний аналіз, де нейронні мережі можуть бути використані для виявлення музичних жанрів, розпізнавання мелодій або навіть створення нової музики.

Такий підхід краще використовувати для глибокого аналізу звуку, але погано підходить для аналізу у реальному часі оскільки нейроні мережі виконують аналіз на основі припущення та не завжди є правдивими, а глибокий аналіз композиції взагалі не потрібен щоб використовувати гітару як контролер, оскільки аналізувати потрібно не аудіо доріжку, а аудіо потік — тобто звук у певний момент часу.

Нейроні мережі можуть гарно справитись з аналізом тембру, що корисно для аналізу звуку з декількома музичними інструментами. Оскільки гітара підключається напряму до комп’ютера, то аналіз тембру не є пріоритетним.

В теорії використання нейронної мережі можливо завдяки використанню бібліотек brainjs або tensorflowjs. Є безліч наборів нот на сервісі huggingface, але оскільки потрібен точний інструмент, то нейроні мережі можна використовувати потім для пошуку акордів або рекомендацій щодо навчання.

2.3. Проблеми при аналізі звуку

WebAudio надає змогу зчитувати з звукового сигналу частоти та магнітуду сигналу. Тому доцільно використовувати алгоритм ACF (Auto-Correlation Function), який використовує автокореляційну функцію для визначення основних частот і здатний виявляти артикуляційні точки в мовленні. ACF може бути використаний для виявлення пікових частот, амплітуди, ширини сигналу і форми сигналу. ACF не дуже підходить для аналізу складних звуків з багатою кількістю нот, наприклад гітарних акордів. Для аналізу більш складного звуку як акорди вже потрібно використовувати техніки машинного навчання, тобто нейрону мережу або використовувати трансформацію Фур'є, за допомогою якої можна отримати хвилі з яких складається аудіо хвиля. Перед використанням трансформації Фур'є потрібно зробити оригінальну хвилю гладкою, щоб видали шуми.

Але для розв'язання задач цієї роботи достатньо використовувати пошук домінантної ноти.

Для гарного аналізу звуку потрібно використовувати чистий звук гітари без ефектів, тому ефекти потрібно буде додавати потім у вебдодатку. Оскільки програма не використовується у професійному звукозаписі, то можна додавати звукові прості ефекти.

Для аналізу звуку гітари використовуються алгоритми на основі аналізу звукових хвиль. Браузерне API WebAudio надає інструменти для отримання даних зі звукового потоку.

3. РОЗРОБКА АЛГОРИТМІВ ІНТЕРПРЕТАЦІЇ СИГНАЛІВ ГІТАРИ

3.1. Вибір інструментів для розробки бібліотеки

Для розробки бібліотеки для використання гітари як контролера було обрано мову програмування TypeScript, яка може компілюватися у мову JavaScript. Мова TypeScript надає можливість визначати типи даних для змінних, функцій та об'єктів, що допомагає виявляти помилки під час розробки на етапі компіляції, а не під час виконання. Це сприяє створенню більш надійних та стабільних бібліотек. Також строга типізація дає підказки користувачам, які використовують бібліотеку, адже вони завжди знають що роблять конкретні функції чи класи бібліотеки. Оскільки бібліотека має використовуватися у браузері, її обов’язково потрібно перетворити у JavaScript. Для цього використовується система збірки Vite. Vite - це надбудова над системами збірки EsBuild та Rollup. EsBuild швидко компілює TypeScript до JavaScript, а Rollup керує процесом збірки бібліотеки.

3.2. Архітектура бібліотеки

Бібліотеку можна поділити на 3 модулі:

- Analyzer — виконує аналіз звукового потоку;
- AMP — виконує трансформацію звукового потоку та відтворює його;
- Recorder — записує звуковий потік у аудіофайл.

Усі модулі користуються браузерним інтерфейсом WebAudio для доступу до звукового потоку з мікрофона. Діаграму пакетів бібліотеки представлено на рисунку 3.1.

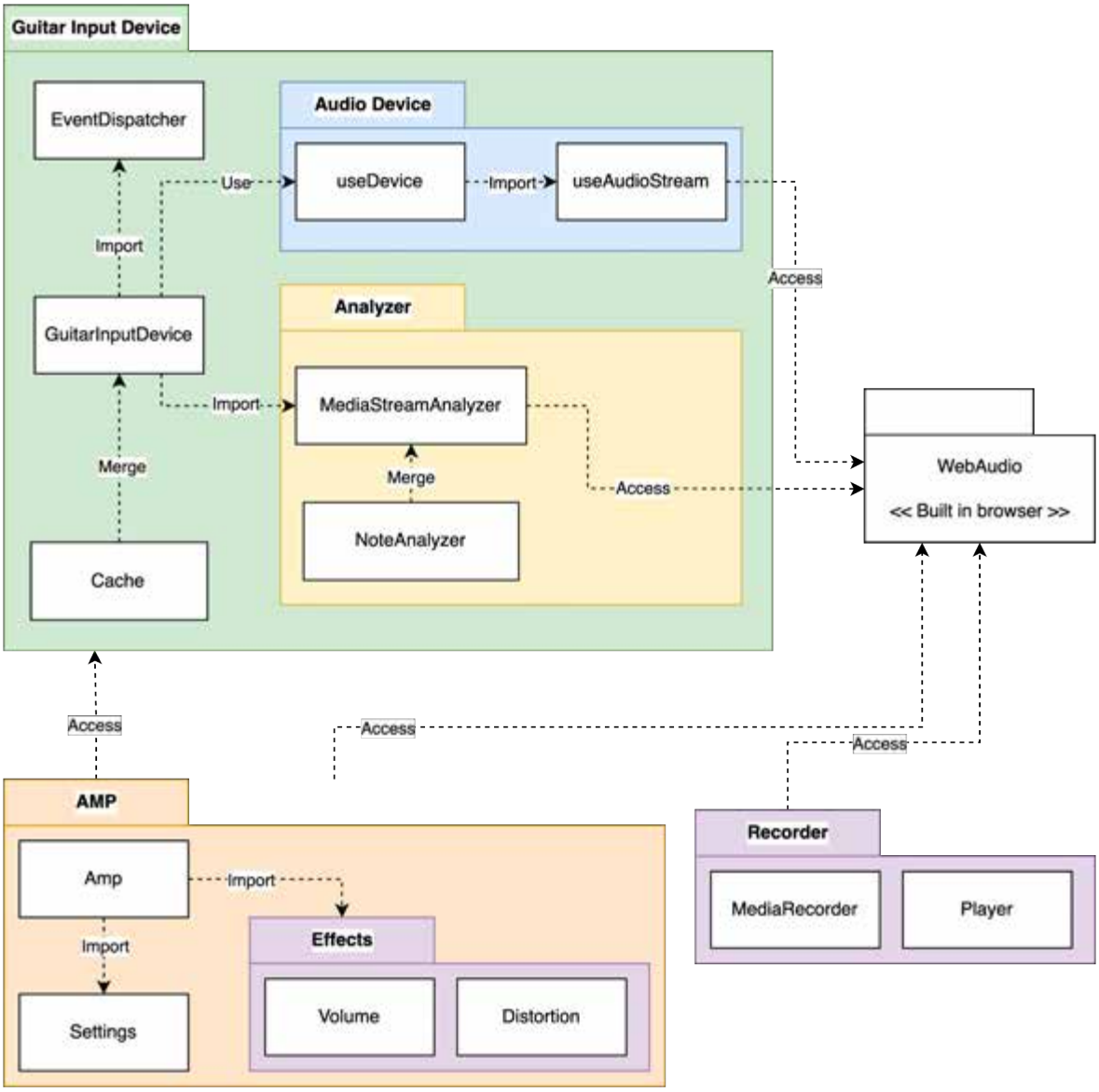

Рисунок 3.1 - діаграма пакетів бібліотеки

Модуль Guitar Input Device складається з таких частин:

- EventDispatcher - спеціальний клас, який додає методи addEventListener, щоб інтерфейс Guitar Input Device був схожим на інтерфейс взаємодії з браузерним програмним інтерфейсом;

- AudioDevice - пакет, який через WebAudio отримує доступ до мікрофона та створює звуковий потік;
- Analyzer - модуль, який оброблює звуковий потік та повертає зіграну ноту, октаву, частоту та гучність кожні 16мс.

Кожні 16мс. після обробки звукового сигналу клас GuitarInputDeivce створює подію “input” з параметрами проаналізованого звуку.

Створюючи новий об’єкт класу GuitarInputDevice, користувач може під'єднати до нього аудіо потік. Клас використовує кеш, щоб не обробляти один потік декілька раз.

Модуль AMP включає такі частини:

- AMP - компонент, який відтворює звук на вебсторінці;
- Effects - набір гітарних ефектів, які змінюють звук;
- Settings - компонент, який регулює налаштування ефектів.
- Модуль Recorder складається з двох компонентів:
- Recorder - записує звуковий потік;
- Player - відтворює записаний потік.

3.3. Принцип роботи аналізатора звуку

Браузерний інтерфейс WebAudio надає інструмент для отримання звукової хвилі у поточний момент часу у вигляді масиву Кожне значення масиву є вибіркою, величиною сигналу в конкретний момент часу. Вміст такого масиву представлено на рисунку 3.2.

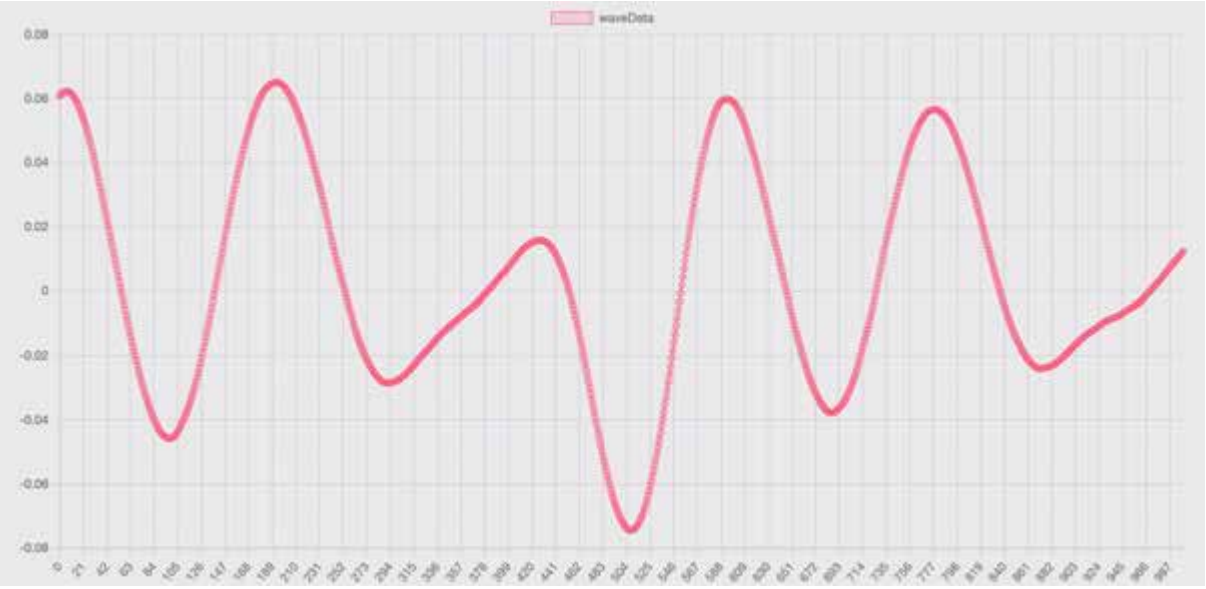

Рисунок 3.2 - вміст масиву з вибіркою звукового сигналу

Виконуючи аналіз такої звукової хвилі можна виявити домінанту частоту звукової хвилі. Для пошуку домінантної частоти виконується автокореляція. Потрібно для кожного “x” на у сигналі знайти суму добутків між “x” та його зміщенням на “x”. Добуток буде показувати наскільки подібні значення сигналу на відстані “x” одне від одного - простіше кажучи наскільки синусоїди схожі при їх зміщенні. Найбільше число у масиві кореляції показує місце де синусоїди схожі якщо їх здвинути на індекс цього числа — цей індекс і буде частотою з якою синусоїда повторюється. Під час пошуку найбільшого с потрібно ігнорувати перші елементи масиву, які спадають і вести пошук починаючи з росту графіку.

```
export function getMaxIndex (numbers: number[]) {
  let d = 0; while (numbers[d] > numbers[d + 1]) d++;

  let max = -Number.MAX_SAFE_INTEGER;
  let maxIndex = -1;
  for (let i = d; i < numbers.length; i++) {
    if (numbers[i] > max) {
      max = numbers[i];
      maxIndex = i;
    }
  }
  return maxIndex;
}

export function acf(buffer: Float32Array, rate: number) {
  const SIZE = buffer.length;

  let c: number[] = new Array(SIZE).fill(0);
  for (let i = 0; i < SIZE; i++)
    for (let j = 0; j < SIZE - i; j++)
      c[i] = c[i] + buffer[j] * buffer[j + i];

  const maxPosition = getMaxIndex(c);

  return rate / maxPosition;
}
```

Рисунок 3.3 - функція пошуку домінантної ноти з звукової доріжки.

Графік кореляції представлено на рисунку 3.4. Індекс найбільшого числа другого піку буде точкою де сигнал починається повторюватися. Найбільший пік вказує на величину затримки часу на якому кореляція набуває найбільшого значення. Щоб знайти частоту у герцах потрібно поділити частоту дискретизації сигналу (sample rate) на затримку. Така частота вказує на те, скільки циклів або періодів сигналу повторюється за одну секунду.

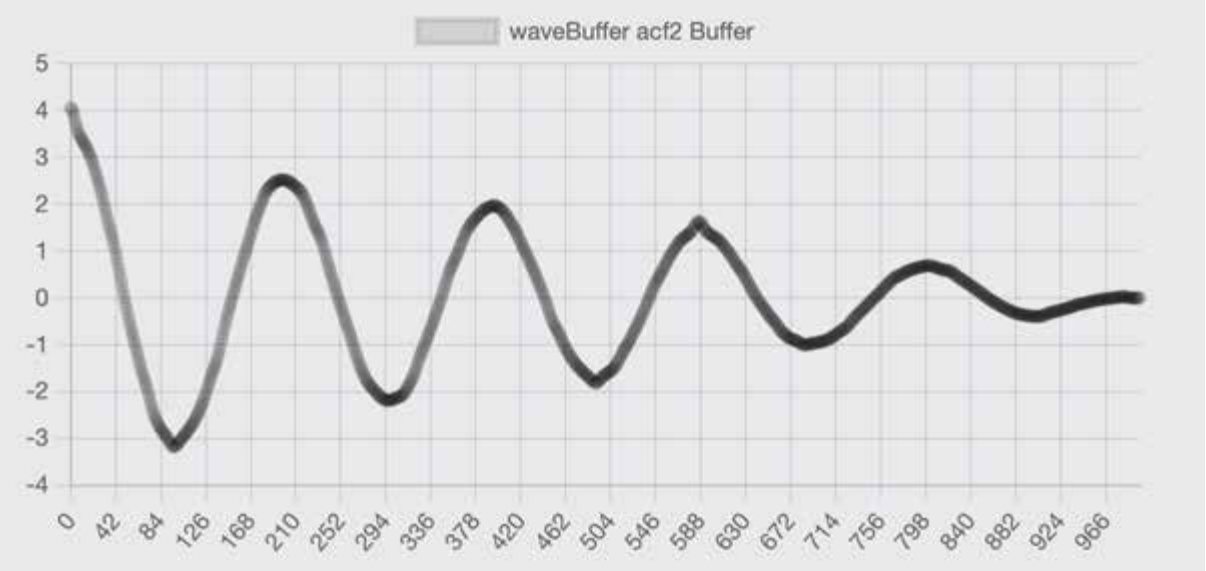

Рисунок 3.4 - графік кореляції для ноти E (мі) 82ГЦ.

Оскільки обробка звуку виконується у реальному часі та теоретично може виконуватися на слабких приладах, як мобільні телефони, то алгоритм можна оптимізувати та прибрати зайве створення масиву та кількість циклів. Поточна реалізація має складність O(n^2 + n). Замість побудови масиву кореляції, зразу шукається індекс найбільшого значення кореляції, що допомагає позбутися зайвого циклу та створення масиву. Оптимізований алгоритм представлено на рисунку 3.5.

```
export function acf2(buffer: Float32Array, rate: number) {
  const SIZE = buffer.length;

  let maxC = -Number.MAX_SAFE_INTEGER;
  let prevC = Number.MAX_SAFE_INTEGER;
  let maxPosition = 0;

  for (let i = 0; i < SIZE; i++) {
    let c = 0;
    for (let j = 0; j < SIZE - i; j++) {
      c = c + buffer[j] * buffer[j + i];
    }

    if (c < prevC) {
      prevC = c;
      continue;
    }

    if (c > maxC) {
      maxC = c;
      maxPosition = i;
    }
  }

  return rate / maxPosition;
}
```

Рисунок 3.5 - Оптимізований алгоритм ACF

Порівняння швидкості алгоритму можна перевірити використовуючи сервіс JsBench.me. Результати представлено на рисунку 3.6. З тестів видно що оптимізований алгоритм майже у 2 рази швидший.

Рисунок 3.6 - порівняння швидкості виконання алгоритму пошуку домінуючої частоти

Для отримання ноти з частоти використовується алгоритм переводу частоти у номер ноти у форматі Musical Instrument Digital Interface (MIDI). Так, зараз усі частоти відлічуються від ноти A, 440HZ або 69 нота MIDI. Щоб дізнатися MIDI індекс потрібно дізнатися в скільки разів частота поточного звуку відрізняється від стандартної настроєної частоти 440 Гц. Оскільки октави поділені на 12, то нам потрібно помножити отримане число на 12. Оскільки кожен музичний полутон (інтервал між сусідніми клавішами на піаніно) відповідає збільшенню частоти в два рази, то логарифм по основі 2 дозволяє точно відобразити ці інтервали на числовій шкалі. До давши номер MIDI можна змістити числовий ряд до ноти A. На рисунку 3.7 показано алгоритм пошуку назви та октави ноти.

```
export const noteNames = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"] as const;

export type Note = typeof noteNames[number]
export type Octave = 0 | 1 | 2 | 3 | 4 | 5 | 6

export const getNoteMIDIIndex = (frequency: number) => {
  // A: 440hz, 69 MIDI note
  return Math.round(12 * (Math.log(frequency / 440) / Math.log(2))) + 69;
}

const getNoteOctave = (note: number) => Math.floor(note / 12) - 1 as Octave

export const getNoteFromBuffer = (buffer: Float32Array, rate: number) => {
  const frequency = acf(buffer, rate);

  // Assume standard tuning
  // Guitar can not play up E6 (1333hz) in 24fret guitar. Skip noise.
  // Guitar can not play below E2 (82hz) in 24fret guitar. Skip noise.
  if (frequency > 1400 || frequency < 70) {
    return {
      note: null,
      octave: null,
      frequency: null,
    }
  }

  const midiNoteIndex = getNoteMIDIIndex(frequency);

  return {
    note: noteNames[getNoteMIDIIndex(frequency) % 12],
    octave: getNoteOctave(midiNoteIndex),
    frequency
  };
}
```

Рисунок 3.7 - алгоритм пошуку назви ноти

Алгоритми пошуку ноти використовуються у класі MediaStreamAnalyzer, який кожного оновлення екрана комп’ютера аналізує поточний звук у MediaStream, що дозволяє отримати актуальну зіграну ноту як найшвидше. Після аналізу він створює подію input.

GuitarInputDevice це спеціальний клас, який створю MediaStreamAnalyzer для звукового потоку у випадку, якщо його немає. Якщо MediaStreamAnalyzer вже був створений, то він береться з кешу. GuitarInputDevice створює таку саму

подію як MediaStreamAnalyzer. Таким чином для одного звукового потоку аналіз відбувається лише раз у відрізок часу.

```
1   import { EventDispatcher } from './EventDispatcher';
2   import { MediaStreamAnalyzer } from './analyzer/MediaStreamAnalyzer';
3   import { GuitarInputEvent } from './types';
4
    You, 40 minutes ago | 2 authors (m0ksem and others)
5   export class GuitarInputDevice extends EventDispatcher<{ 'input': (e: GuitarInputEvent) => void
6     private static cache = new Map<MediaStream, MediaStreamAnalyzer>()
7
8     private streamAnalyzer: MediaStreamAnalyzer | null = null
9
10    private onInput(e: GuitarInputEvent) {
11      this.dispatchEvent('input', [e])
12    }
13
14    connect(stream: MediaStream) {
15      if (this.streamAnalyzer) {
16        this.streamAnalyzer.removeEventListener('input', this.onInput)
17      }
18
19      if (GuitarInputDevice.cache.has(stream)) {
20        this.streamAnalyzer = GuitarInputDevice.cache.get(stream)!
21      } else if (typeof AudioContext !== 'undefined') {
22        this.streamAnalyzer = new MediaStreamAnalyzer(stream)
23        GuitarInputDevice.cache.set(stream, this.streamAnalyzer)
24      } else {
25        // Do nothing in SSR
26        return
27      }
28
29      this.streamAnalyzer.addEventListener('input', (e) => this.onInput(e))
30    }
31
32    get analyzer() {
33      return this.streamAnalyzer
34    }
35
36    stop() {
37      if (this.streamAnalyzer) {
38        this.streamAnalyzer.removeEventListener('input', this.onInput)
39        this.streamAnalyzer.stop()
40      }
41    }
42  }
```

Рисунок 3.8 - реалізація класу GuitarInputDevice

Важливим моментом є перевірка на наявність браузерного API у поточній середі. Оскільки код може виконуватися на сервері, то ми не оброблюємо звукові хвилі, оскільки на сервері не відбувається взаємодія з користувачем та його

мікрофоном. Такі перевірки запобігають помилкам через нестачу браузерного API на сервері під час виконання коду в SSR.

Для пошуку гучності виконується простий пошук амплітуди звукової хвилі.

Використовуючи getFloatTimeDomainData метод з WebAudio AnalyzerNode можна отримувати дані звукової хвилі з звукового потоку. Звуковий потік зберігається у об’єкті MediaStream.

```
You, 41 minutes ago | 2 authors (m0ksem and others)
import { EventDispatcher } from "../EventDispatcher";
import { getNoteFromBuffer } from "./getNote";
import type { GuitarInputEvent } from "../types";
import { useAudioAnalyzer } from "./useAudioAnalyzer";

You, 41 minutes ago | 2 authors (m0ksem and others)
export class MediaStreamAnalyzer extends EventDispatcher<{ 'input': (e: GuitarInputEvent) => void }> {
  private RAFIndex = 0

  public readonly waveBuffer: Float32Array

  constructor(stream: MediaStream) {
    super()
    const { analyser, audioCtx } = useAudioAnalyzer(stream)

    this.waveBuffer = new Float32Array(analyser.fftSize);

    const tick = () => {
      this.RAFIndex = requestAnimationFrame(tick)

      analyser.getFloatTimeDomainData(this.waveBuffer)

      const volume = this.waveBuffer.reduce((a, b) => a > b ? a : b, 0) * 100

      const { note, octave, frequency } = getNoteFromBuffer(this.waveBuffer, audioCtx.sampleRate)

      this.dispatchEvent('input', [{
        note, octave, volume, frequency
      }])
    }

    tick()
  }

  stop() {
    cancelAnimationFrame(this.RAFIndex)
  }
}
```

Рисунок 3.9 - код класу MediaStreamAnalyzer

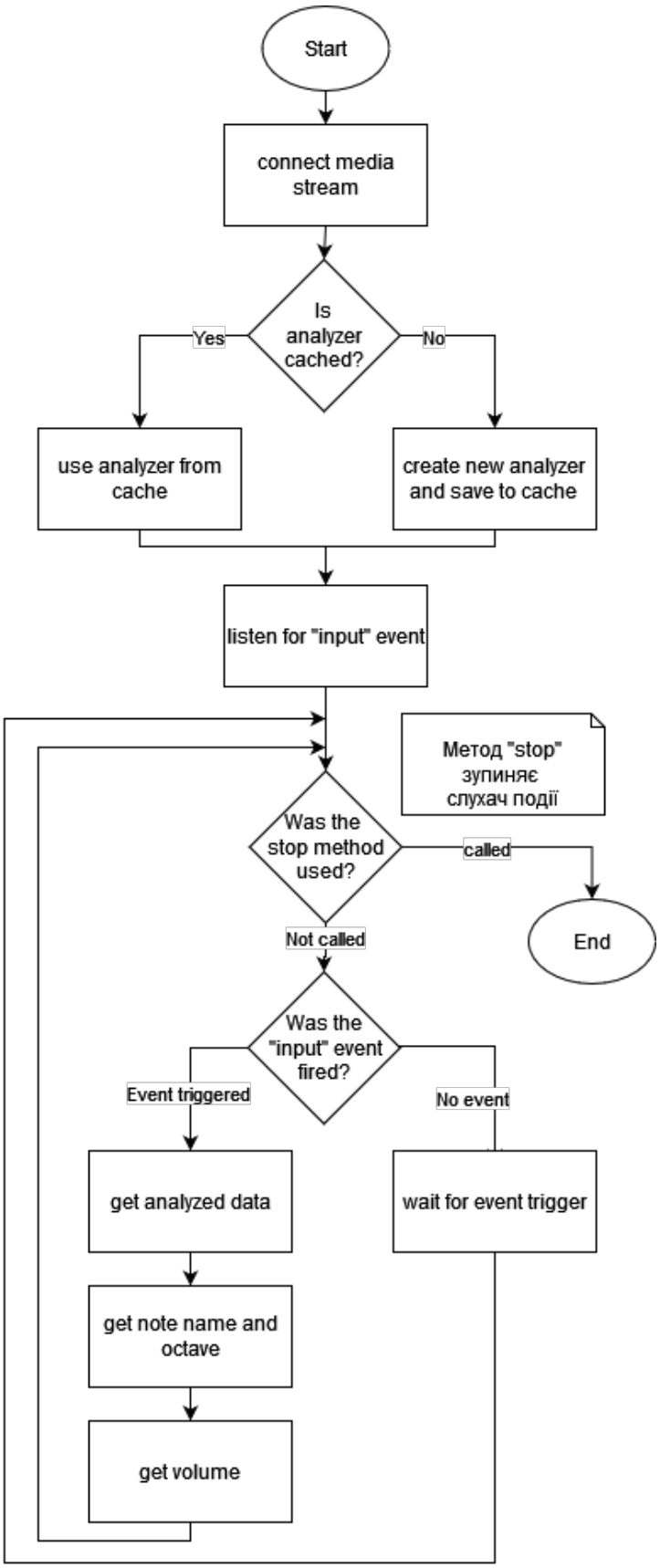

Рисунок 3.10 - Алгоритм роботи Guitar Input Device

3.4. Принцип роботи гітарних ефектів

Для відтворення гітарного звуку з звукового потоку використовується HTML елемент audio, тому модуль AMP (Guitar Amplifier) не може бути простою JavaScript функцією. Модуль AMP представлено у вигляді компонента, створеного з використанням Javascript фреймворку VueJS. Компонент VueJS надає можливість додавати HTML на сторінки з додатковою інтерактивністю з власною системою реактивності. Компоненти з використання VueJS легко компілюються у Web Components - браузерний стандарт, який дозволяє створювати власні HTML елементи без використання додаткових бібліотек чи bundler-ів. Компонент має атрибути (або props) через які можна передавати дані до компонента. Так компонент має такі параметри:

- stream - об’єкт класу MediaStream, який є поточним звуковим потоком;
- volume - числове значення гучності з якою відтворюється звуковий потік у браузері;
- distortion - значення накладання ефекту distortion.

Додаткові ефекти можна створити використовуючи JavaScript бібліотеку ToneJS, наприклад, delay, chorus, wah тощо.

Імплементація компонента можна побачити на рисунку 3.11.

Перед відтворенням звуку він перетворюється у моно сигнал, оскільки деякі гітарні процесори можуть видавати звук в один канал. Потім підключаючи AudioNode можна змінювати звук, змінюючи його гучність або звукову хвилю.

```vue
<template>
  <audio ref="audio" />
</template>

<script setup lang="ts">
import { PropType, watchEffect } from 'vue';
import { createDistortion } from '../effects/distortion';
import { createVolume } from '../effects/volume';

const props = defineProps({
  stream: { type: [Object, null] as PropType<MediaStream | null>, required: true },
  volume: { type: Number, default: 1 },
  distortion: { type: Number, default: 0 },
})

let context = new AudioContext()

watchEffect(() => {
  if (props.stream) {
    if (context) {
      context.close()
      context = new AudioContext()
    }

    const inLine = new MediaStreamAudioSourceNode(context, { mediaStream: props.stream });
    const mono = new ChannelSplitterNode(context, { numberOfOutputs: 2, channelInterpretation: 'discrete' });
    inLine.channelInterpretation = 'speakers'

    let line = inLine
      .connect(mono)
      .connect(createVolume(context, toRef(props, 'volume')))

    if (props.distortion > 0) {
      line = line.connect(createDistortion(context, toRef(props, 'distortion')))
    }

    line.connect(context.destination)
  }
});
</script>
```

Рисунок 3.11 - компонент відтворення звуку

Для перевірки покращення аналізу гітарного звуку рекомендується використовувати гітарний звук без додавання ефектів, тому бібліотека надає можливість додавати власні ефекти вже після аналізу. Такий спосіб не підходить для професійного запису звука, але для професійного запису звука краще використовувати спеціальне обладнання на студії. Найпопулярніший ефект, який створив самий великий вплив на використання гітари у сучасній музиці це ефект Distortion. Для його імплементації потрібно взяти оригінальну (чисту) звукову хвилю та обрізати її піки. Приклад представлено на рисунках 3.12 та

3.13, де перший це чиста звукова хвиля до накладання ефекту, а другий з накладанням ефекту, який закруглює пікові частоти.

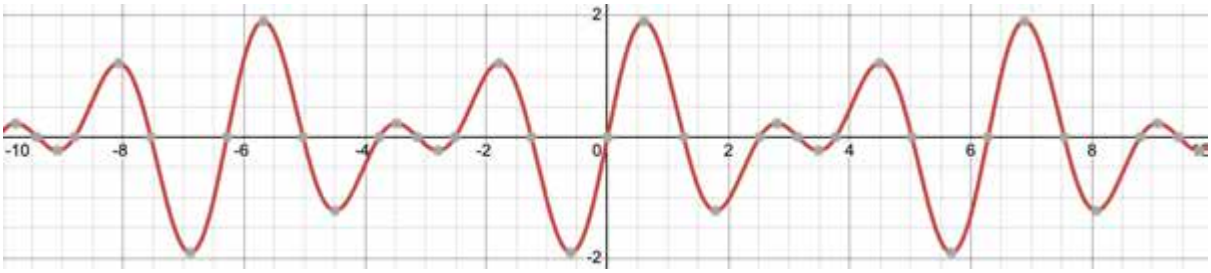

Рисунок 3.12 - чиста звукова хвиля з частотами 2гц та 3гц

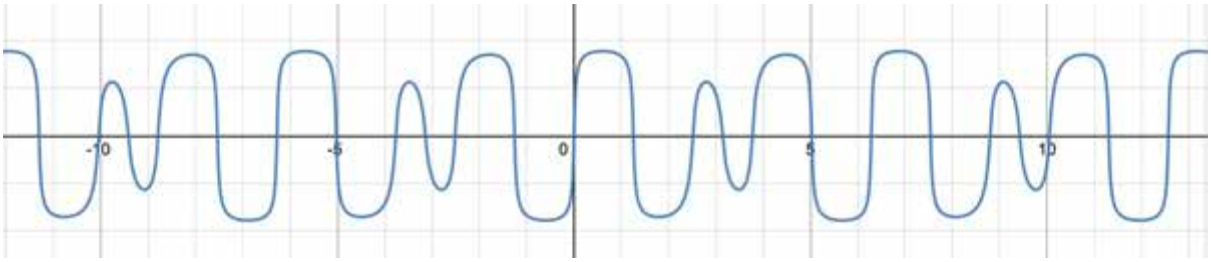

Рисунок 3.13 - звукова хвиля з використанням ефекту Distortion з частотами 2 Гц та 3 Гц

Формула для зміни звукової хвилі представлена на рисунку 3.14. Для зміни початкової хвилі, потрібно накласти на неї більш пологу функцію з закругленими піками. Замість sin(2x)+sin(3x) можна підставити будь-яку частоту. Формула виведена вручну для відтворення кращого звуку. Можна змінювати константи для зміни вихідного звуку. Так, у коді потім константи можна буде змінювати і користувач зможе керувати силою ефекту.

$$y = \frac{(3 + 120) \cdot \left(\sin(2x) + \sin(3x)\right) \cdot \frac{\pi}{10}}{20 \cdot \left|\sin(2x) + \sin(3x)\right| + \pi}$$

Рисунок 3.14 - формула для створення кривої ефекту Distortion

Для того, щоб накласти такий ефект на практиці у коді потрібно використовувати WebAudio API і WaveSharpenNode, створити масив з хвилею за формулою та передати його до WaveSharpenNode, який потім використовувати у ланцюжку ефектів. Приклад коду представлено на рисунку 3.15.

```
1   import { Ref } from 'vue'
2
3   function makeDistortionCurve(context: AudioContext, amount: number) {
4     const gain = 100 * amount;
5     const sampleRate = context.sampleRate;
6     const curve = new Float32Array(sampleRate);
7     const deg = Math.PI / 120;
8     let x;
9
10    for (let i = 0; i < sampleRate; ++i) {
11      x = i * 2 / sampleRate - 1;
12      curve[i] = (((3 + gain) * x * 20 * deg / (Math.PI + gain * Math.abs(x)))) / 20;
13    }
14
15    return curve;
16  }
17
18  export const createDistortion = (context: AudioContext, amount: Ref<number>) => {
19    const distortion = new WaveShaperNode(context, {
20      oversample: '2x',
21      curve: makeDistortionCurve(context, amount.value),
22    })
23
24    watchEffect(() => {
25      distortion.curve = makeDistortionCurve(context, amount.value)
26    })
27
28    return distortion
29  }
```

Рисунок 3.15 - код, відповідаючий за створення ефекту

3.5. Вибір інструментів для розробки

Очевидно, що для того, щоб розробляти продукти для веббраузера необхідно використовувати мови HTML, CSS та JavaScript. Однак, для побудови бібліотек краще використовувати мову TypeSciprt замість JavaScript. Мова TypeScript надає додатковий синтаксис для опису типів, які використовуються у коді та компілюється у JavaScript. Це дає користувачеві бібліотеки можливість використовувати бібліотеку та мати уявлення про типи, які потребують функції та класи. Типи дають підказки у середовищі розробки та надають можливість показати помилку при неправильному використанні класів та функцій з бібліотеки. Очевидно, що такий підхід сприяє покращенню взаємодії з бібліотекою. Також, під час розробки бібліотеки використовується фреймворк VueJS, для створення компонента відтворення звуку. VueJS - це прогресивний фреймворк, який має власну систему реактивності, рендерінгу та можливість компіляції у Web Components - стандарт для браузера, який дозволяє створювати використовувати власні HTML елементи. Оскільки архітектура VueJS опирається на композицію компонентів, бібліотека експортує спеціальні функції (composables), якими можна розширяти інші компоненти. Для простого способу написання таких функцій використовується бібліотека з готовими інструментами vueuse.

Для збірки бібліотеки використовується інструмент збірки проєктів vite. Для написання тестів використовується бібліотека vitest. Для візуального тестування використовується бібліотека vue-chartjs для побудови графіків функцій звукової хвилі.

Для демонстраційних програм використовувався фреймворк nuxtjs, який розширяє можливості vuejs завдяки автоматичним імпортам та побудові сторінок

базуючись на файлах. Для швидкого написання стилів використовувався інструмент unocss, який значно спрощує написання CSS коду надаючи готові стильові рішення. Щоб створювати елементи інтерфейсу, які підтримують керування з клавіатури було використано бібліотеку vuestic-ui з великою кількістю готових елементів інтерфейсу у вигляді компонентів.

Для написання коду використовувалась IDE Visual Studio Code.

3.6. Практична реалізація розробленого методу

Для тестування та демонстрації можливостей бібліотеки було розроблено веб застосунок де гітара використовується як гітарний контролер.

Застосунок має наступні вебсторінки:

- Тренування — сторінка з можливістю запису власного виконання;
- Список пісень — сторінка з списком пісень, навігація по списку може виконуватися лише використовуючи гітару;
- Програвач — табулатура з маркером, який показує коли потрібно грати ноти.
- Для побудови вебзастосунку використовується вебфреймворк VueJS та бібліотека готових компонентів VuesticUI. Інтерфейс поділений на 4 частини:
- Інформаційна — поточна нота, її октава, частота та гучність;
- Налаштування — комбо бокс для вибору мікрофона, слайдер для регулювання гучності відтворення звуку та сили ефекту Disturtion;
- Запис — інструменти для запису та прослуховування записаного звуку;

- Табулатуру — відображення зіграних нот у вигляді гітарної табулатури.

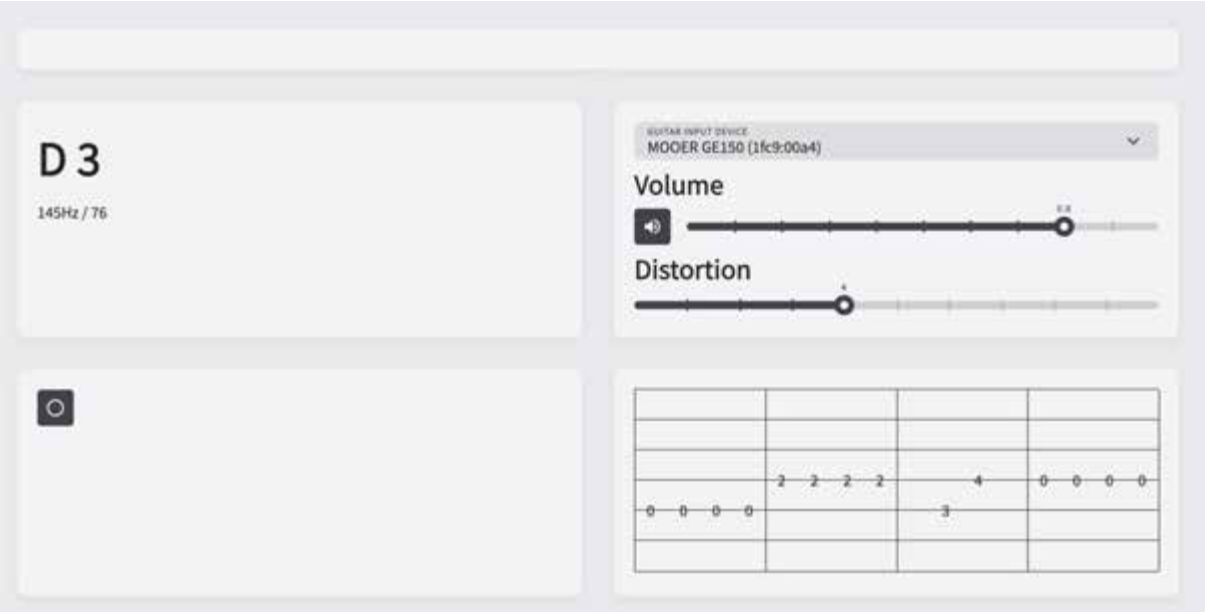

Рисунок 3.16 - інтерфейс тренувальної сторінки

Для сторінки з списком композицій, доступних до виконання. Кожен елементу списку це кнопка, натиснувши на яку можна переглянути додаткову інформацію про композицію та вибрати складність виконання. Зверху списку є пошукова смуга. Для такого інтерфейсу реалізоване керування з гітари та клавіатури. При взаємодії з пошуковим полем спливає віртуальна клавіатура для набору тексту з гітари. Така програма використовує ноти A, C, D, F# як переміщення вверх, вліво, вправо та вниз. Завдяки специфічному місцеперебуванню інтерфейс подібний до стрілок на клавіатурі. Гітара у стандартному строї на 24 ладах має 49 унікальних нот, але якщо змінювати гучність, то кількість можна збільшити до 147 потенційних клавіш. Для імітації клавіш “стрілок” потрібно вибрати які ноти на грифі гітарі, які будуть відповідати за кнопки “верх”, “вниз”, “вліво” та “вправо”. Для прикладу такі позиції можна вибрати як показано на рисунку 3.13.

Рисунок 3.17 - місця де потрібно зажати струну, щоб імітувати клавіатуру

Імітуючи клавіатуру та маючи інтерфейс, яким можна керувати з клавіатури можна керувати усім інтерфейсом. Так достатньо імітувати кнопки: tab, shift + tab, arrow-left, arrow-right, arrow-down, arrow-up, enter, space.

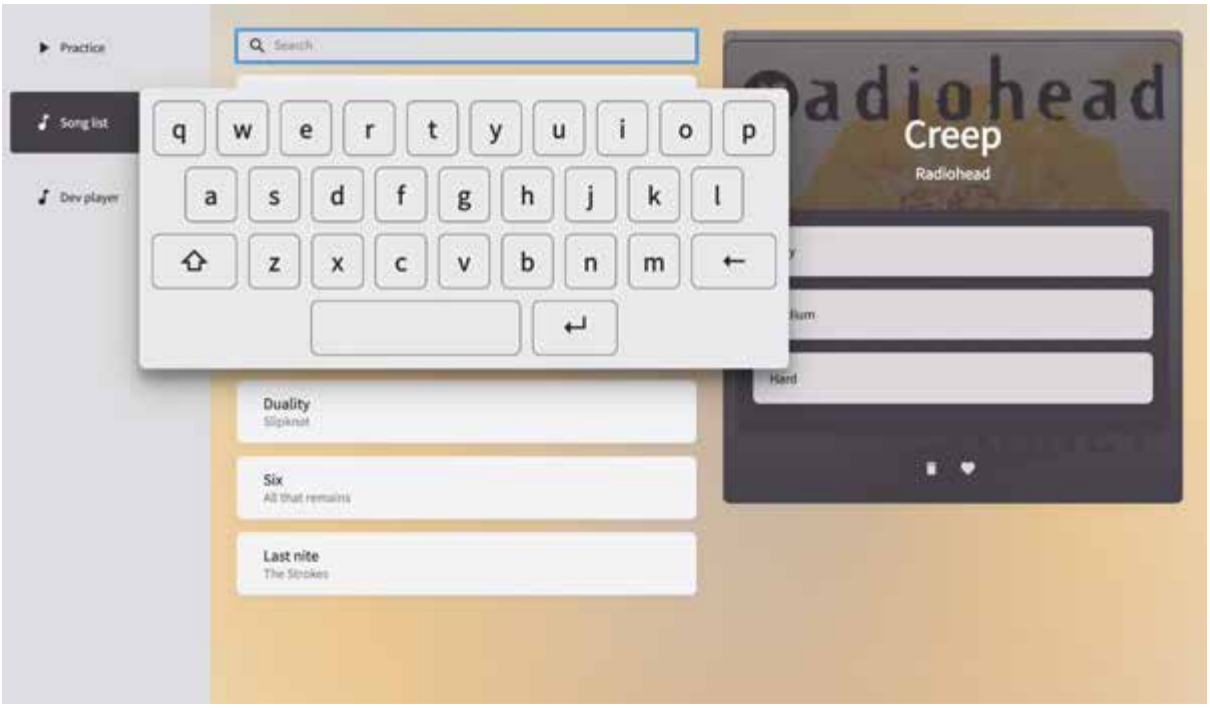

Рисунок 3.18 - інтерфейс сторінки як прикладу користування інтерфейсом за допомогою гітари

Так інтерфейси схожі на інтерфейси ігрових консолей, які не мають клавіатури, а контролюються спеціальним контролером, який називається “геймпад”.

Для того, щоб повністю розкрити потенціал гітари як приладу вводу, було розроблено демонстраційну сторінку, де потрібно виконувати коректну ноту на гітарі у певний час, таким чином вивчаючи композицію.

Рисунок 3.19 - інтерфейс виконання композиції з миттєвою перевіркою коректності у вигляді табулатури

4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1. Потенційна сфера застосування

Найбільше використання гітари як контролера у вебзастосунках підходить саме для навчання грі на гітарі. Хоча й кількість потенційних сигналів, які виробляє гітара більше за кількість клавіш на клавіатурі, через складність їх виконання зменшується швидкість створення таких сигналів, а отже швидкість вводу інформації, тому гітару як контролер краще використовувати саме як гітару. Для вводу тексту як з клавіатури на гітарі потрібно створити звуковий сигнал, а для цього потрібно дві руки одразу — одна, щоб зажати потрібну ноту, а другою відтворити звук, щоб створити звукову хвилю за допомогою струни гітари. Але, наприклад, для запису табулатури легше виконати конкретну ноту на гітарі ніж записати її руками. Хоча є проблема того що табулатуру можна записати максимум до п’ятого ладу. Проблема існує, тому що у гітари кожні п’ять ладів і вищу струну ноти повторюються. Наприклад, нота на п’ятому ладу шостої струни такої ж самої частоти як і відкрита п’ята струна. Таким чином на десяти ладах гітари зіграна у двох місцях і зрозуміти в якому саме неможливо. Тому насправді для запису точної табулатури такий підхід також не дуже точний, хоча й набагато швидший ніж набирати на клавіатурі.

Найкраще розроблений метод підходить для перевірки правильності виконання ноти у певний відрізок часу або робити підказки на основі вже зіграних нот.

4.2. Алгоритм створення гітарного тренажера

На основі розробленої бібліотеки дуже просто створювати тренажери для гри на гітарі. За розробниками та дизайнерами стоїть задачі вибрати найкращій

візуальний інтерфейс тренажера, вибрати які ноти потрібно зіграти та перевіряти правильність слухаючи проаналізовані ноти з бібліотеки. Декілька таких інтерфейсів представлено на рисунках 3.15, 4.1, 4.2.

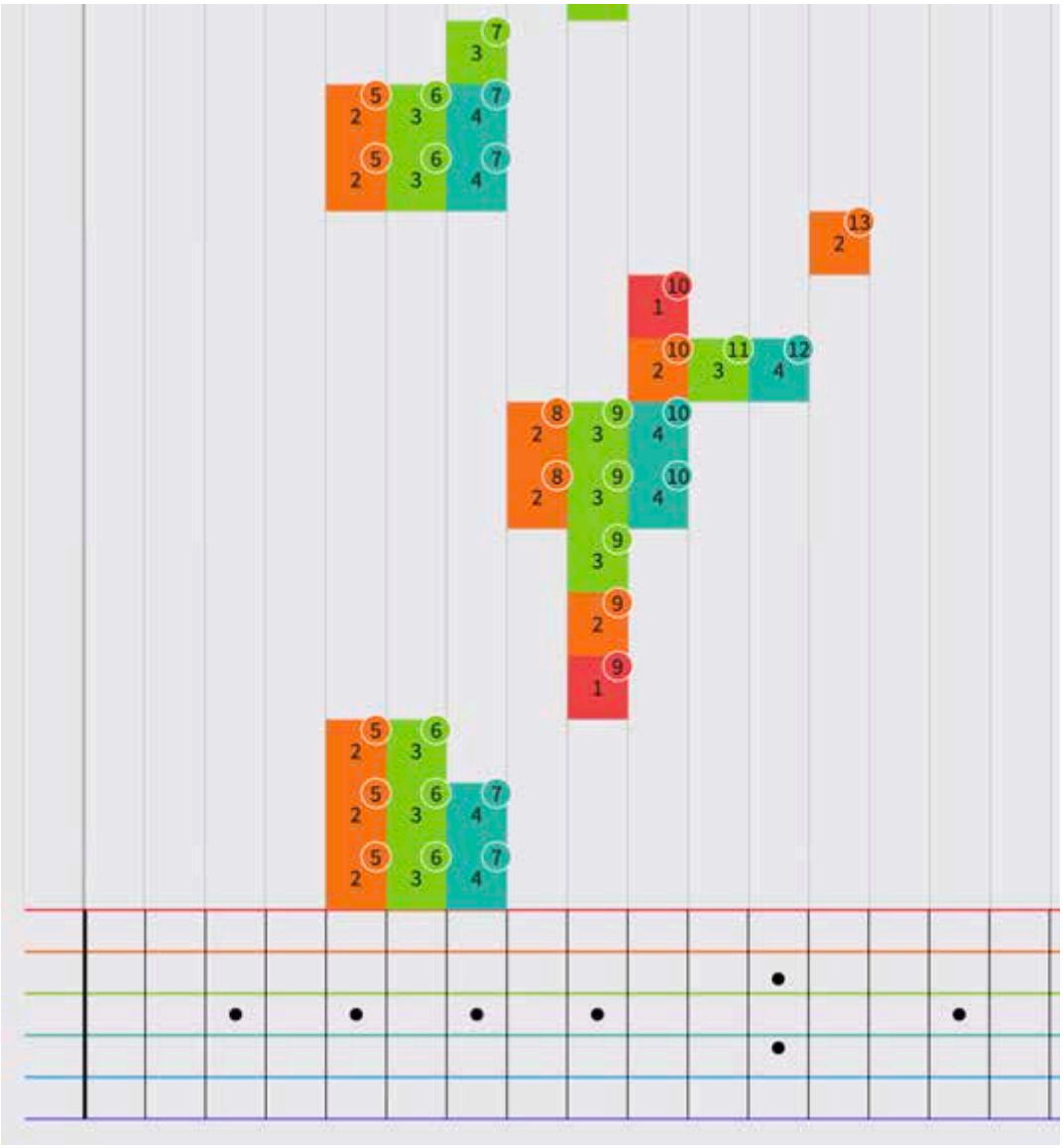

Рисунок 4.1 - інтерфейс тренажера у вигляді “дощу” з падаючими нотами на гриф гітари

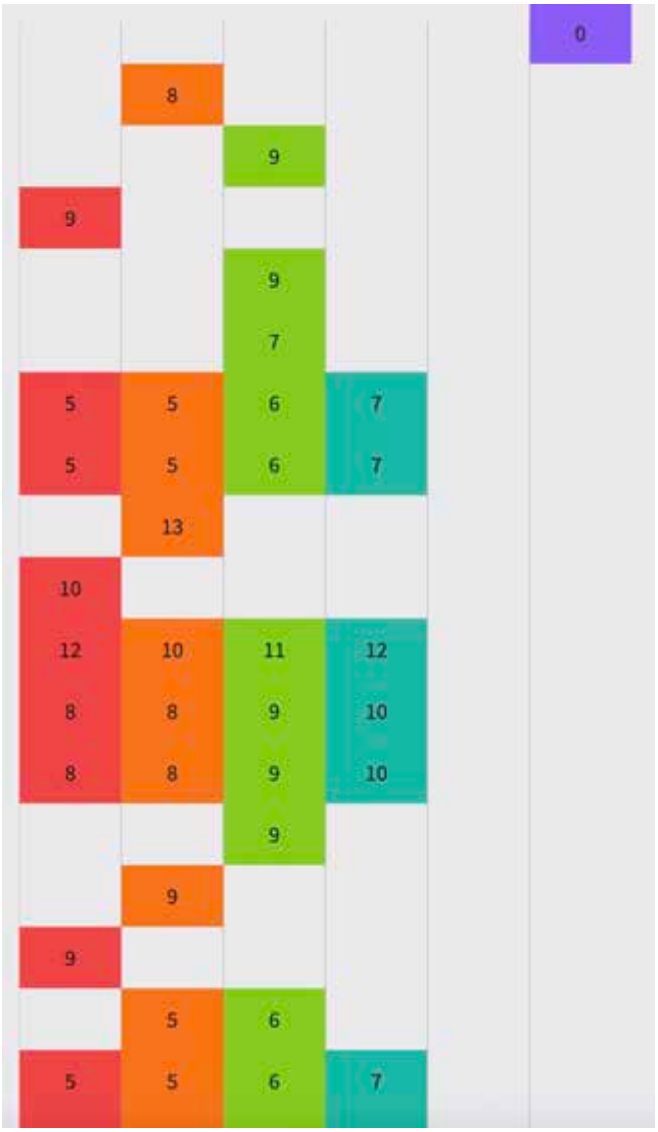

Рисунок 4.2 - інтерфейс у вигляді “дощу” без гітарного грифа

Для імплементації тренажера потрібно виявити час на виконання однієї ноти. Зазвичай композиції мають темп, який записується у бітах за хвилину (Beats per minute - BPM). Таким чином за формулою t = 60 / BPM можна виявити скільки секунд виконується один біт (або одна нота). Таким чином потрібно слухати усі зіграні ноти за проміжок часу t і перевіряти чи зіграно саме потрібну ноту у цей проміжок часу. Оскільки бібліотека може оброблювати поточний звук частіше, ніж ноти потрібно грати, то доцільно оброблювати масив зіграних нот у

період коли їх потрібно грати. Для цього слід використовувати такі параметри як: частота появи ноти у масиві та гучність нот. Дослідивши такий об’єм даних можна точніше сказати яку ноту виконав музикант. Наприклад, на рисунку 4.3 показано графік де:

- y — частота зіграних нот;
- x — час, коли потрібно зіграти один біт композиції (тобто одну ноту);
- Яскравість синього кольору — наскільки часто нота є в одному біті;
- Яскравість червоного кольору — наскільки гучна нота.

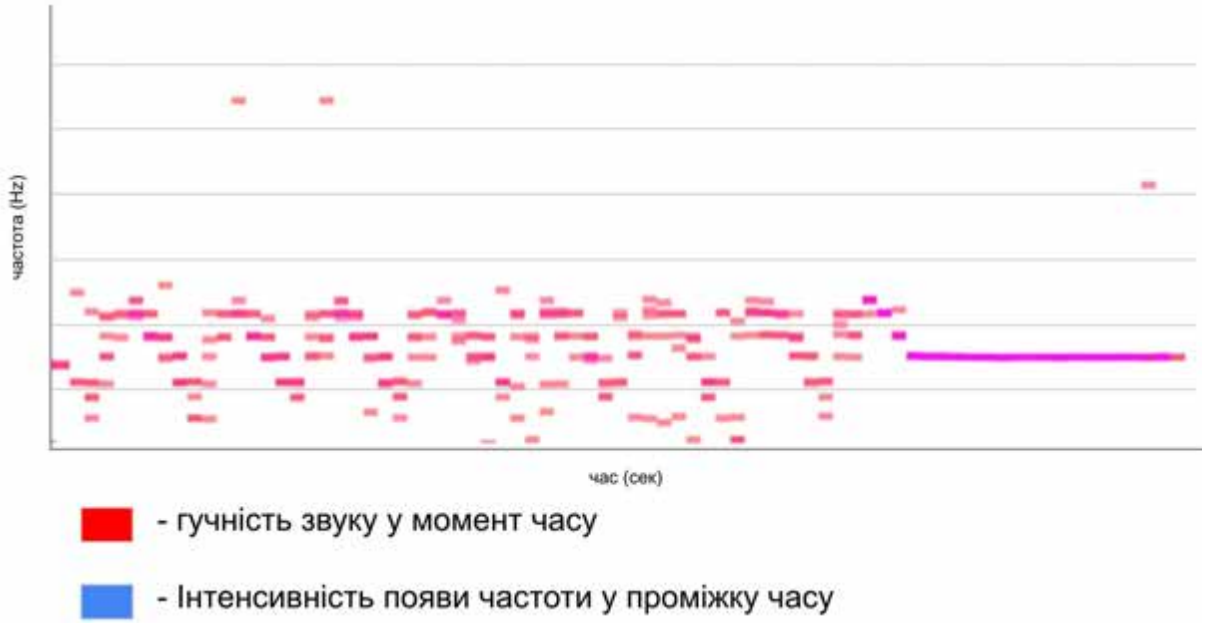

Рисунок 4.3 - графік щільності та гучності нот.

Таким чином можна побачити, що хоч користувач і задів можливо інші струни, але не на довго та не гучно. З графіку видно які саме ноти домінують в одному біті й виявити яку насправді користувач намагався виконати при швидких темпах і складних композиціях.

Перевіривши зіграну ноту з потрібною, можна вивести користувачеві результат на екран.

ВИСНОВКИ

За період роботи над магістерською роботою було проведено аналіз сучасних способів побудови вебзастосунків. В основному було досліджено сферу роботи зі звуком у веббраузері, зокрема способи виявлення ноти зіграної на гітарі. Для розв'язання задачі аналізу гітарного звуку у середовищі веббраузера було обрано оптимальний алгоритм, а саме алгоритм, який базується на кореляції. Виявлено що даний алгоритм є достатньо швидким та точним для аналізу гітари у реальному часі. Також було виявлено недоліки алгоритму, та виявлено для яких цілей і сфер підходить це рішення.

З використанням запропонованого рішення було створено демонстраційні вебзастосунки для підтвердження ефективності та корисності рішення. Запропонований алгоритм можна використовувати у вигляді JavaScript бібліотеки для побудови вебзастосунків спрямованих на перевірку коректності виконання композиції, запису виконаних нот чи керуванням графічним інтерфейсом використовуючи гітару як контролер у веборієнтованих вебзастосунках. Таким чином можна зробити навчання грі на гітарі цікавішим та корисним для користувачів. Для бібліотеки обрано зручний програмний інтерфейс, який надає розробникам систему основану на подіях для повного контролю введених даних з гітари, у вигляді композиційних функцій для швидкого впровадження функціонала у вебзастосунок та компонентів для використання інструментів відтворення звуку у вигляді HTML елементів.

Протягом тестування розробленої системи, було перевірено коректність та точність роботи алгоритмів і доцільність розробки запропонованих методів для задоволення потреб користувачів.

Результати дослідження можна використовувати для подальшої побудови вебзастосунків для навчання грі на гітарі. Запропоноване рішення можна

розширити з використанням нейронних мереж та технологій Data Mining для створення рекомендацій та персональних уроків таким чином покращивши досвід користувача у взаємодії з гітарою під час навчання.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Wolfe J. Note names, MIDI numbers and frequencies [Електронний ресурс] / Joe Wolfe – Режим доступу до ресурсу: https://newt.phys.unsw.edu.au/jw/notes.html.
2. Woody Herman. DSPFuzz A Guitar Distortion Pedal Using The Stanford DSP Sheild [Електронний ресурс] / Woody Herman – Режим доступу до ресурсу: https://web.stanford.edu/class/ee264/projects/EE264_w2015_final_project_herman.pdf.
3. A brief history of computer music. [Електронний ресурс] – Режим доступу до ресурсу: https://www.musicradar.com/news/tech/a-brief-history-of-computer-music-177299.
4. Develop WebAudio Plugins in a Web Browser [Електронний ресурс] / Shihong Ren, Stéphane Letz, ann Orlarey, та ін.] – Режим доступу до ресурсу: https://ccrma.stanford.edu/~rmichon/publications/doc/WAC-19-plugin.pdf.
5. Josh C. Waveforms [Електронний ресурс] / Comeau Josh. – 2018. – Режим доступу до ресурсу: https://pudding.cool/2018/02/waveforms/
6. Buffa M. MT5: a HTML5 multitrack player for musicians [Електронний ресурс] / M. Buffa, A. Hallili, P. Renevier – Режим доступу до ресурсу: https://inria.hal.science/hal-01150455/document.
7. Web Audio API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
8. Thibauld Nion. Autocorrelation functions [Електронний ресурс] / Thibauld Nion. – 2012. – Режим доступу до ресурсу: https://etudes.tibonihoo.net/literate_musing/autocorrelations.html.
9. Grant S. But what is the Fourier Transform? A visual introduction. [Електронний ресурс] / S. Grant, W. River – Режим доступу до ресурсу: https://www.3blue1brown.com/lessons/fourier-transforms.

10. Miguel M. Naive Bayes Classifier in JavaScript [Електронний ресурс] / Mota Miguel. – 2015. – Режим доступу до ресурсу: https://miguelmota.com/blog/naive-bayes-classifier-in-javascript
11. Ridhi T. Predictive Analysis for Risk Reduction in Data Mining [Електронний ресурс] / Thakur Ridhi. – 2019. – Режим доступу до ресурсу: https://www.researchgate.net/publication/334961363_Predictive_Analysis_for_Risk_Reduction_in_Data_Mining
12. Raghav A. Music Genre Classification with Machine Learning [Електронний ресурс] / Agrawal Raghav. – 2022. – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2022/03/music-genre-classification-project-using-machine-learning-techniques
13. Створення додатку Vue [Електронний ресурс] – Режим доступу до ресурсу: https://ua.vuejs.org/guide/essentials/application.html
14. Building a JavaScript guitar pedalboard [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.trysmudford.com/blog/pedalboard/.
15. Sebastian T. Autocorrelation [Електронний ресурс] / Taylor Sebastian – Режим доступу до ресурсу: https://corporatefinanceinstitute.com/resources/data-science/autocorrelation/#:~:text=Autocorrelation%20refers%20to%20the%20degree,also%20known%20as%20serial%20correlation
16. Gilbertson D. A Web Audio experiment: detecting piano notes [Електронний ресурс] / David Gilbertson – Режим доступу до ресурсу: https://david-gilbertson.medium.com/a-web-audio-experiment-666743e16679.
17. What Is Aliasing In Audio? The Science, Sound, And How To Avoid It [Електронний ресурс] – Режим доступу до ресурсу: https://audiosorcerer.com/post/what-is-aliasing-in-audio/.
18. Fotis Adamakis·. 1. VueUse - The library that makes Vue 3 worth the upgrade [Електронний ресурс] / Fotis Adamakis· – Режим доступу до ресурсу:

https://medium.com/js-dojo/vueuse-the-library-that-makes-vue-3-worth-the-upgrade-7047c5bb00ce.

19. Madhu Balakrishna. Introduction to Web Audio API [Електронний ресурс] / Madhu Balakrishna – Режим доступу до ресурсу: https://dyte.io/blog/web-audio-api/.
20. Sam Dutton. Get started with WebRTC [Електронний ресурс] / Sam Dutton – Режим доступу до ресурсу: https://web.dev/articles/webrtc-basics.
21. Cameron Adams. How to develop a Typescript Library [Електронний ресурс] / Cameron Adams – Режим доступу до ресурсу: https://medium.com/@cameronadams1225/how-to-develop-a-typescript-library-ade8d329636.
22. TypeScript: JavaScript With Syntax For Types [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/.
23. Abiola Farounbi. Getting started with Vue composables [Електронний ресурс] / Abiola Farounbi – Режим доступу до ресурсу: https://blog.logrocket.com/getting-started-vue-composables/.
24. Використання додатків для дистанційного навчання [Електронний ресурс] – Режим доступу до ресурсу: https://prometheanworld.com.ua/vykorystannya-dodatkiv-dlya-dystantsijnogo-navchannya/.
25. Iskander Samatov. Write an audio visualizer from scratch with vanilla JavaScript [Електронний ресурс] / Iskander Samatov – Режим доступу до ресурсу: https://blog.logrocket.com/audio-visualizer-from-scratch-javascript/.
26. Visualizations with Web Audio API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API.