Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

MIDI веб-контролер для дистанційного керування

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІз-21-1
Антон ВІТКОВСЬКИЙ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Артем ГУК
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Вітковському Антону Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи MIDI веб-контролер для дистанційного керування

затверджена наказом по університету від " 05 " травня 2025 р. № 43 Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 року

3. Вхідні дані до роботи

1) протокол передачі даних: MIDI;
2) протокол з'єднання: USB;
3) Інтерфейс: Web

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір технологій розробки та інструментальних засобів;
3) розробка програмних модулів;
4) тестування та відлагодження;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	06.05.25-09.05.25	
2	Вибір технології розробки та	10.05.25-13.05.25	
	інструментальних засобів		
3	Розробка алгоритмічного забезпечення	14.05.25-17.05.25	
4	Розробка програмних модулів	19.05.25-28.05.25	
5	Відлагодження програмних модулів	29.05.25-05.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
7	Подання кваліфікаційної роботи керівникові та	10.06.25-11.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	12.06.25-13.06.25	
	вання		

Дата видачі завдання " 05 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Артем ГУК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 97 с., 22 рис., 3 табл., 1 дод., 10 джерел.

MIDI, USB, WEB MIDI API, MUSIC SEQUENCER, DAW, JAVASCRIPT, CSS, WEB.

Метою кваліфікаційної роботи є створення веб застосунку, що забезпечує керування музичними пристроями за допомогою протоколу MIDI для передачі музичної інформації.

У рамках роботи було розроблено веб MIDI контролер, що дозволяє створювати до 16 музичних послідовностей, з підтримкою 8 пристроїв одночасно. Користувач може редагувати ноти, їх інтенсивність, довжину кроків, налаштовувати MIDI-канали та вибирати вихідні пристрої. Архітектура застосунку побудована на основі подієвої моделі, з окремими модулями для зберігання даних, взаємодії з MIDI-інтерфейсом, графічного інтерфейсу користувача та синхронізації через вбудований метроном. Проект реалізовано з використанням HTML, CSS, JavaScript, Web MIDI API та Node.js як локального сервера.

ABSTRACT

Bachelor’s thesis: 97 pages, 22 figures, 3 tables, 1 appendices, 10 sources.

MIDI, USB, WEB MIDI API, MUSIC SEQUENCER, DAW, JAVASCRIPT, CSS, WEB.

The purpose of the qualification work is to create a web application that provides control of music devices using the MIDI protocol to transmit music information.

As part of the work, a web-based MIDI controller was developed, allowing users to create up to 16 musical sequences, supporting 8 devices simultaneously. The user can edit notes, their intensity, step length, configure MIDI channels, and select output devices. The architecture of the application is based on an event-based model, with separate modules for data storage, interaction with the MIDI interface, graphical user interface, and synchronization via the built-in metronome. The project was implemented using HTML, CSS, JavaScript, Web MIDI API, and Node.js as a local server.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface – інтерфейс прикладного програмування; набір функцій і протоколів для взаємодії між програмними компонентами.

BPM – Beats Per Minute – одиниця виміру темпу в музиці, яка вказує кількість тактів (ударів) на хвилину.

DAW – Digital Audio Workstation – програмне середовище для запису, редагування, обробки та відтворення музичних композицій.

JSON – JavaScript Object Notation – легкий текстовий формат обміну даними, що використовується для збереження та передачі структурованої інформації.

MIDI – Musical Instrument Digital Interface – цифровий інтерфейс для обміну музичними повідомленнями між електронними музичними інструментами, комп’ютерами та іншими пристроями.

VST – Virtual Studio Technology – технологія плагінів, яка дозволяє інтегрувати в DAW віртуальні інструменти та ефекти.

ВСТУП

Упродовж останніх років цифрова музика стрімко розвивається й охоплює дедалі ширше коло користувачів. Від простих MIDI-файлів сучасні технології перейшли до складних цифрових аудіо станцій (DAW), які дозволяють реалізовувати повноцінні музичні проекти – від запису до фінальної обробки.

Одним із ключових інструментів цифрової музики є протокол MIDI (Musical Instrument Digital Interface)[1], який забезпечує передачу команд між пристроями – як апаратними, так і програмними. Якщо раніше MIDI використовувався переважно у професійних студіях, то нині він активно застосовується в мобільних застосунках, навчальних платформах, інтерактивних інсталяціях та веб-застосунках.

Разом із тим, традиційні DAW мають низку обмежень: вони складні в освоєнні, потребують потужного обладнання, прив’язані до конкретних операційних систем і потребують встановлення. Це створює бар’єр для новачків, студентів або музикантів, які працюють у мобільному або дистанційному форматі.

Web MIDI API[2] – веб-технологія, яка дозволяє працювати з MIDI-пристроями прямо у браузері без додаткового програмного забезпечення. Це відкриває можливість створення простих, доступних та кросплатформених рішень для музичної творчості, зокрема MIDI-контролерів нового покоління.

Мета цієї роботи полягає у створенні веб застосунку для дистанційного керування MIDI-пристроями за допомогою Web MIDI API. Розроблюваний застосунок має реалізувати базовий функціонал керування MIDI-пристроями через веб-інтерфейс, а також запропонувати потенційну платформу для можливостей, які можуть бути додані в майбутньому: генерація MIDI-послідовностей на основі алгоритмів або нейромереж, режим спільної роботи кількох користувачів через мережу, інтерактивна візуалізація нот у

реальному часі та інше.

Для досягнення мети передбачено виконання таких завдань: аналіз існуючих рішень, вивчення протоколу MIDI і специфікації Web MIDI API, вибір технологій, проектування архітектури, реалізація інтерфейсу, підтримка збереження/завантаження проектів, тестування з різними пристроями.

Методологія роботи базується на поєднанні теоретичного аналізу та практичної реалізації. Вивчено технічну документацію, оглянуто аналогічні проекти, реалізовано прототип на JavaScript, побудовано архітектуру, протестовано роботу з реальними пристроями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі буде розглянуто принципи дистанційного керування електронними музичними пристроями за допомогою протоколу MIDI. Спочатку буде проаналізовано сам протокол – його структуру, типи повідомлень та особливості передачі музичної інформації. Далі розглядатиметься робота MIDI через USB як одного з найзручніших сучасних способів підключення, що не потребує додаткових драйверів. Окрему увагу буде приділено наявним програмним і апаратним рішенням для керування MIDI-пристроями – таким як цифрові аудіо станції та фізичні секвенсори – з подальшим аналізом їхніх переваг і обмежень. Це дозволить сформувати обґрунтовану основу для оцінки доцільності використання веб-застосунку на базі Web MIDI API як альтернативного рішення.

1.1 Електронні музичні пристрої

Сучасні електронні музичні пристрої – це синтезатори, MIDI-контролери, драм-машини, семплери та інші пристрої(рисунок 1.1), які підтримують цифрову взаємодію з комп’ютером або один з одним. Ці пристрої здатні не лише генерувати звук, а й приймати команди ззовні, зокрема через MIDI-протокол, що дозволяє дистанційне керування їх параметрами: висотою тону, темпом, гучністю, вибором пресетів, ефектами тощо.

Рисунок 1.1 – Приклад музичних пристроїв

Принцип дистанційного керування полягає в тому, що замість ручної взаємодії з фізичними кнопками або енкодерами, користувач надсилає команди з іншого пристрою або застосунку. Це відкриває широкі можливості для автоматизації, інтеграції в більші системи та створення гнучких музичних сценаріїв. Наприклад, під час живого виступу можна одночасно керувати кількома синтезаторами з одного контролера або навіть з веб-браузера. Таке дистанційне управління особливо актуальне у випадках, коли пристрої фізично віддалені, інтегровані у складні студійні мережі, або використовуються у віддаленому викладанні музики.

1.2 Протокол MIDI через USB

MIDI-пристрої можна підключати до комп’ютера кількома способами. Найпоширеніший – через USB, коли пристрій автоматично розпізнається системою без потреби встановлювати драйвери. Інший варіант – через класичний 5-контактний MIDI-кабель, який потребує зовнішнього MIDI-інтерфейсу. Також можливе бездротове з’єднання через Bluetooth MIDI, зручне для мобільних сценаріїв, але з деякою затримкою. Нарешті, існують віртуальні MIDI-пристрої, які використовуються для маршрутизації сигналів між програмами всередині системи. Web MIDI API дозволяє працювати з

усіма цими типами підключення напряму з браузера.

Рисунок 1.2 – Фізичні 5-контактові MIDI порти

Протокол MIDI (Musical Instrument Digital Interface) був розроблений у 1983 році як стандарт для передачі музичної інформації між електронними інструментами. Його головна перевага полягає в тому, що він не передає сам звук, а лише цифрові команди – наприклад, «натиснута нота C4 з силою 90», або «встановити контроль гучності на 80%». Завдяки цьому MIDI є універсальним і не залежить від конкретного виробника чи типу пристрою. MIDI-повідомлення мають чітку структуру: статусний байт (тип події, номер каналу) та один або два додаткових байти даних (наприклад, номер ноти і її сила)[1]. Це дозволяє з мінімальними витратами передавати музичні події в режимі реального часу. З моменту свого створення MIDI став стандартом у музичній індустрії та пережив кілька етапів розвитку. Останні доповнення, такі як MIDI 2.0, вводять покращену роздільну здатність, нові формати повідомлень, двонаправлену передачу даних та більшу гнучкість. Однак навіть базовий MIDI 1.0 залишається достатнім для більшості задач, зокрема в браузерному середовищі. Наприклад, за допомогою MIDI-команди NOTE_ON (0x90) можна активувати певну ноту на синтезаторі. У поєднанні з Web MIDI API ця команда може надсилатись прямо з JavaScript-коду[2].

Сучасні MIDI-пристрої зазвичай підключаються до комп’ютера через USB. Більшість з них підтримують клас «USB MIDI device» і не потребують

встановлення спеціальних драйверів – операційна система автоматично розпізнає пристрій як MIDI-інтерфейс. Ця універсальність є однією з ключових переваг. На практиці це означає, що веб-застосунок, побудований на основі Web MIDI API, може взаємодіяти з підключеним через USB пристроєм без додаткового налаштування або встановлення програм[2]. Браузер (наприклад, Chrome або Edge) з підтримкою Web MIDI API одразу надає доступ до списку підключених пристроїв[2] і дозволяє відправляти/отримувати повідомлення. Таким чином, взаємодія з MIDI-пристроями в такій архітектурі є максимально простою та доступною для користувача, і не залежить від сторонніх інструментів – лише від наявної реалізації підтримки MIDI в операційній системі та браузері.

1.3 Аналіз існуючих рішень

Розглянемо, які існують програмні та фізичні(апаратні) рішення для роботи з MIDI. Це допоможе зрозуміти, які функції вони пропонують, у чому їхні переваги, а також які можуть бути обмеження.

1.3.1 Програмні рішення

На сьогодні на ринку існує велика кількість програмних рішень для роботи з MIDI. Одна з найвідоміших програм – Ableton Live (рисунок 1.3). Загалом, це потужні цифрові аудіо станції, які мають розширений функціонал, підтримують інтеграцію з MIDI-пристроями, ефекти, багатодоріжкове редагування та синхронізацію[3]. Серед базових функцій вони мають:

- багатодоріжкове MIDI-відтворення та редагування
- інтеграція з VST/AudioUnit плагінами
- складні MIDI-автоматизації
- мікшер, ефекти, маршрутизація сигналу

- повна підтримка MIDI-контролерів
- експорт проектів у студійному якості

Проте в багатьох випадках ці інструменти можуть бути надмірними. По-перше, вони мають складний інтерфейс, що вимагає часу для навчання. По-друге, це комерційні продукти, які потребують встановлення на конкретну ОС. По-третє, вони не завжди зручні для простих сценаріїв – наприклад, швидкого запуску MIDI-команд чи тестування підключення пристрою. Тож серед недоліків можна назвати:

- висока складність інтерфейсу, неінтуїтивне керування для новачків
- значне навантаження на систему, потреба в потужному залізі
- платна ліцензія ($100–1000+)
- встановлення на конкретну ОС, обмежена кросплатформенність
- надмірний функціонал для простих сценаріїв (тест, демонстрація, навчання)

Рисунок 1.3 – Скріншот програми Ableton Live

1.3.2 Апаратні рішення

Фізичні секвенсори (наприклад Korg SQ-1) також активно використовуються у професійних студіях. Вони мають надійну побудову, автономну роботу, та мають такий функціонал[4]:

- фізичний контроль над керуванням музичною інформацією через тактильний інтерфейс
- створення, запис і відтворення MIDI-патернів у реальному часі, без зупинки для налаштовування
- автономна робота без комп’ютера
- пряме підключення до синтезаторів або модульних систем

Проте їх вартість висока, функціональність обмежена фіксованими елементами управління, а налаштування не завжди гнучке, і мають певні інші недоліки:

- висока ціна ($300–1000+)
- обмежений набір контролерів (наприклад, 8 або 16 треків)
- складність у зміні конфігурації або розширенні функцій
- відсутність гнучкого візуального інтерфейсу
- потреба в окремому живленні та фізичному доступі

Рисунок 1.4 – Апаратний MIDI секвенсор KORG SQ-1

1.4.3 Веб рішення

У цьому контексті браузерне рішення на основі Web MIDI API має деякі переваги: воно безкоштовне, кросплатформенне та не вимагає встановлення. Крім того, воно зручне для дистанційного використання – у навчанні, експериментах, репетиціях, де важлива швидкість та простота доступу до інструменту. Але веб додаток має і певні потенційні обмеження:

- не всі браузери підтримують Web MIDI API (наприклад, Safari – частково підтримує цей стандарт)[5]
- обмеження безпеки(потрібне налаштування дозволу від користувача)
- впирається на реалізацію Web MIDI стандарту
- а сам MIDI функціонал залежить від реалізації MIDI у ОС – якщо операційна система або драйвер некоректно розпізнає пристрій, Web MIDI API не зможе з ним працювати
- збереження даних часто реалізується у вигляді JSON, без гнучких форматів проекту

1.3.4 Порівняння та висновок

Хоча професійні DAW та апаратні секвенсори є потужними інструментами, вони не завжди зручні або доступні для швидких, мобільних або освітніх сценаріїв. У таких випадках веб-застосунок на основі Web MIDI API має очевидні переваги – простота, швидкість запуску, універсальність, гнучкість інтерфейсу та повна відсутність залежності від ОС чи встановленого ПЗ. Це робить його перспективним інструментом для сучасного MIDI-контролю.

Таблиця порівняння всіх типів рішень(Таблиця 1.1) допомагає візуально обґрунтувати доцільність використання саме веб-рішення як легкого, адаптивного та універсального інструменту для роботи з MIDI-пристроями, особливо в освітньому, дослідницькому або мобільному середовищі.

Таблиця 1.1 – Порівняння характеристик існуючих рішень.

Характеристика	ПО рішення	Апаратні рішення	Веб-застосунок
Тип інтерфейсу	На рівні зваємодії з ОС	Кнопки, енкодери, обмежені екрани	Веб інтерфейс
Складність	Висока	Середня	Низька
Крос-платформеність	Часткова (залежить від ОС)	Ні (залежать від апаратної форми)	Так (будь-який сучасний браузер)
Вартість	Висока ($100–500+)	Дуже висока ($300–1000+)	Безкоштовно
Інсталяція	Потрібна	Необхідне обладнання	Не потрібна (працює з браузера)
Можливість віддаленої роботи	Обмежено (через налаштування)	Взагалі відсутня	Повна підтримка
Ідеальний сценарій використання	Студійне виробництво	Живі виступи, автономна робота	Навчання, тестування, швидкі сесії, онлайн

2 ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

У цьому розділі розглядаються програмні та апаратні засоби, які було використано для реалізації веб-застосунку MIDI контролеру. Вибір технологій базується на вимогах до кросплатформенності, доступності через браузер, інтеграції з фізичними музичними пристроями та підтримки роботи в реальному часі. Детально аналізуються особливості протоколу MIDI, структура його повідомлень, принципи обробки подій та багатоканальності. Окрему увагу приділено Web MIDI API як основному інтерфейсу для взаємодії з пристроями, а також опису апаратного забезпечення, використаного для тестування функціональності застосунку.

2.1 Програмні засоби розробки та середовище

Основною мовою програмування є JavaScript яка забезпечить всю логіку роботи застосунку: зчитування MIDI-подій, надсилання команд, керування інтерфейсом користувача, а також взаємодію з Web MIDI API. Сам Web MIDI API буде ключовим компонентом, оскільки саме він забезпечує доступ до MIDI-портів, дозволяє виявляти підключені пристрої, надсилати повідомлення, а також реагувати на події в реальному часі.

Структура та візуальне оформлення інтерфейсу будуть реалізовані за допомогою HTML та CSS. Інтерфейс буде розроблено таким чином, щоб він був зрозумілим, адаптивним та функціональним – з кнопками, слайдерами, індикаторами, панеллю пристроїв та візуалізацією MIDI-подій. Це дасть змогу створити середовище, зручне як для демонстрації, так і для практичного використання.

Для розгортання веб-застосунку буде використано Node.js як серверна частина. Він виконує роль простого веб-сервера, який обслуговує HTML, CSS та JavaScript-файли, а також забезпечує локальний запуск застосунку під

час розробки.

Для тестування та налагодження буде використан браузер Google Chrome. Його було обрано як основне середовище розробки завдяки повній підтримці Web MIDI API, стабільній роботі з MIDI-пристроями та вбудованим інструментам розробника (Chrome DevTools). DevTools дозволяють спостерігати вхідні MIDI-повідомлення, перевіряти передачу даних, аналізувати продуктивність застосунку та взаємодію з DOM-елементами інтерфейсу.

Код проекту буде організований за допомогою Git. Це дасть можливість відслідковувати зміни, створювати окремі гілки для експериментів, зберігати резервні копії та потенційно ділитися кодом з іншими учасниками або викладачами.

У якості редактора буде Visual Studio Code. Це зручне середовище розробки з підтримкою розширень для веб-технологій, автодоповненням синтаксису, вбудованим терміналом і можливістю швидкого запуску локального сервера.

2.2 Специфікація протоколу MIDI та властивості нот

Протокол MIDI (Musical Instrument Digital Interface) був розроблений як уніфікований спосіб комунікації між електронними музичними інструментами. Його ключова ідея – передавати не аудіо, а команди, які керують тим, як має звучати музика[1]. Це робить MIDI надзвичайно ефективним у передачі складних музичних подій з мінімальною затримкою.

Стандарт MIDI передбачає 16 незалежних каналів, які функціонують паралельно в межах одного MIDI-з’єднання. Кожен канал може використовуватись для керування окремим інструментом або окремим шаром одного багатоканального пристрою. Наприклад, канал 1 може відповідати за синтезатор, канал 2 – за драм-машину, канал 16 – за мікшерний пульт студії. Це дозволяє створювати багатошарові композиції, передаючи різні музичні

партії по різних каналах одночасно.

Поканальність сигналу також спрощує маршрутизацію в складних музичних мережах, де один комп'ютер або контролер може взаємодіяти з кількома зовнішніми пристроями через один MIDI-інтерфейс(рисунок 2.1). Завдяки цьому можна надсилати команди до конкретного пристрою, не заважаючи іншим. У контексті Web MIDI API це дозволяє веб-застосунку призначати MIDI події окремим каналам, керувати різними пристроями одночасно, або розподіляти функціональність одного пристрою за кількома логічними ролями.

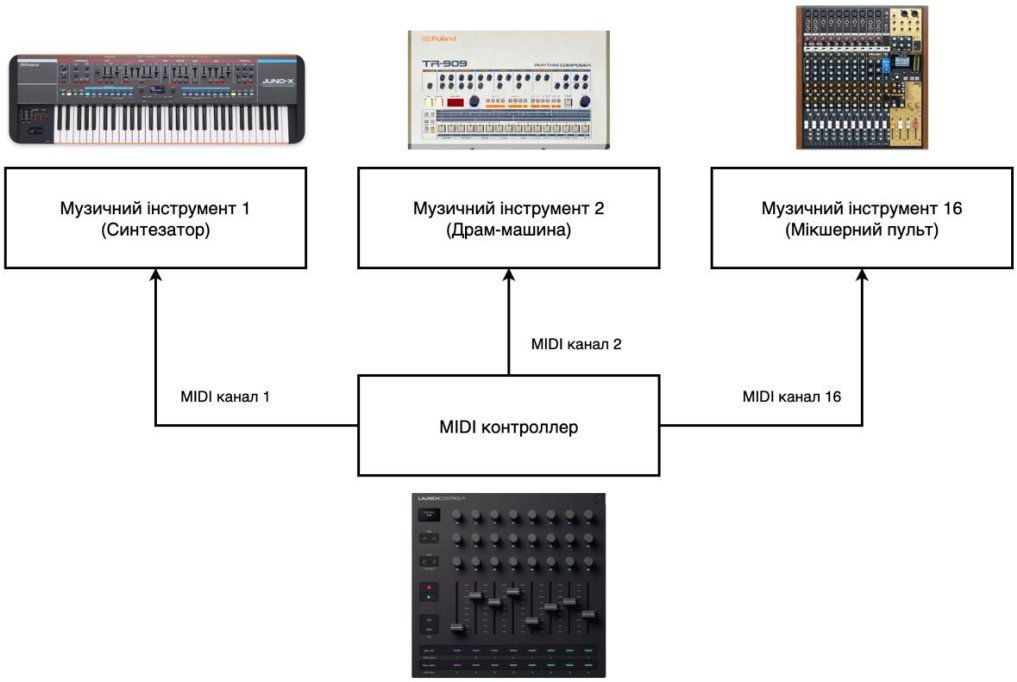

Рисунок 2.1 – схема 3-канальної MIDI мережі

MIDI працює на основі подій, де кожне повідомлення є окремою одиницею дії – наприклад, натискання чи відпускання ноти, зміна параметру або запуск синхронізації програвання. Ці повідомлення передаються в послідовному порядку з точною часовою міткою (timestamp)[1], що дає змогу зберігати точність у ритмі та синхронізації, навіть при великій кількості подій, відтворювати точний темп, тривалість нот, одночасних акордів та ритмічних структур. У класичному апаратному MIDI дані передаються зі швидкістю 31,25 кбіт/с, чого достатньо для більшості музичних задач,

зокрема для надсилання десятків подій на секунду. У сучасних реалізаціях через USB або Web MIDI API швидкість передачі значно вища і практично не має затримок для користувача, що дозволяє використовувати MIDI в реальному часі – як у студійній роботі, так і у веб-застосунках, де потрібна моментальна реакція на дії користувача.

Повідомлення у MIDI мають компактний формат – здебільшого складаються з 2 або 3 байтів(таблиця 2.1). Перший байт є статусним і містить тип події (наприклад, Note On, Note Off, Control Change) разом із номером каналу, а наступні байти несуть дані – наприклад, номер ноти та її інтенсивність (velocity). Така мінімальна структура дозволяє передавати велику кількість подій з дуже низьким навантаженням на систему, що особливо важливо в реальному часі, коли навіть мілісекундні затримки можуть бути відчутними. Завдяки цьому MIDI залишається ефективним і стабільним навіть у складних музичних конфігураціях із великою кількістю одночасних сигналів.

Таблиця 2.1 – Структура MIDI повідомлення

Статусний байт								Байт даних 1								Байт даних 2							
1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	1	0	0	1	1	0
Тип повідом.				Номер каналу				Додаткові дані повідомлення								Додаткові дані повідомлення							

Статусний байт (Status Byte) – це перший байт будь-якого MIDI-повідомлення. Його структура:

- 1-й біт завжди 1, що позначає, що це статусний байт.
- наступні 3 біти визначають тип повідомлення (наприклад, Note On, Note Off, Control Change).
- останні 4 біти вказують номер каналу (від 0 до 15), що дозволяє передавати повідомлення на 16 MIDI-каналах.

Після статусного байту йдуть один або два байти з даними. У кожному

байті:

- 1-й біт завжди 0, що відрізняє їх від статусних байтів.
- інші 7 бітів містять числове значення (від 0 до 127).

Кількість байтів даних залежить від типу повідомлення.

MIDI повідомлення бувають різних типів, але нотні MIDI-повідомлення використовуються для того, щоб вмикати та вимикати ноти на електронних музичних інструментах[1], таких як синтезатори, клавіатури або віртуальні інструменти в комп'ютері. Існує два основні типи нотних повідомлень:

- Note On – увімкнення ноти – надсилається, коли музикант натискає клавішу на MIDI-клавіатурі, і містить інформацію про те, яку ноту потрібно заграти та з якою динамікою. У деяких випадках замість Note Off використовується Note On із силою натискання 0 – це стандартний прийом для економії даних.

- Note Off – вимкнення ноти – надсилається, коли клавішу відпускають, і припиняє звучання ноти.

Кожна нота в MIDI має такі властивості(рисунок 1.6):

- Note Number - номер ноти в діапазоні від 1 до 127(наприклад, 60 - це нота "До" 4 октави(C4), 61 - "До-дієз" 4 октави(D#4)і так далі)

- MIDI channel - номер MIDI каналу(1-16), що вказує, до якого інструмента належить нота

- Velocity - динаміка ноти(наскільки голосно або інтенсивною вона повинна звучати, наприклад сила натискання клавіші на фортепіано)

- Duration - активна фаза ноти, скільки нота триває(час між повідомленням On і повідомленням Off)

Приклади MIDI повідомлень:

Note On (натискання ноти C4):

- 1) Статусний байт: 0x90 (Note On, канал 1),
- 2) Номер ноти: 0x3C (C4 = 60),
- 3) Velocity: 0x64 (гучність, наприклад 100)

- У байтах: 0x90 0x3C 0x64
- У десятковому форматі: 144 60 100

Note Off (відпускання ноти C4):

- 1) Статусний байт: 0x80 (Note Off, канал 1),
- 2) Номер ноти: 0x3C (C4 = 60),
- 3) Velocity: 0x00 (немає сили натискання)
- У байтах: 0x80 0x3C 0x00
- У десятковому форматі: 128 60 0

Ці властивості та характеристики важливі для нашого застосунку(рисунок 2.2), бо це дає нам розуміння та технологічні деталі концепції самого алгоритму, основу для графічного відображення і користувацького інтерфейсу. Саме з урахуванням параметрів ноти – таких як висота, гучність, тривалість та канал – формується логіка побудови музичних послідовностей, візуалізація нот у часі, реакція інтерфейсу на дії користувача, та точне відображення подій у взаємодії з підключеними пристроями в реальному часі.

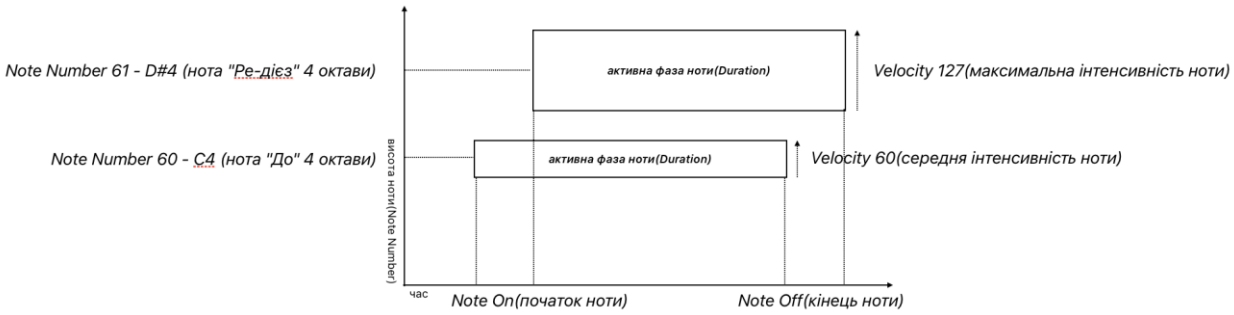

Рисунок 2.2 – Властивости MIDI нот у часі

2.3 Web MIDI API

Web MIDI API – це специфікація, яка дозволяє веб-застосункам взаємодіяти з MIDI-пристроями у реальному часі напряму з браузера. Вперше підтримана у Google Chrome, вона відкрила можливість створення повноцінних музичних інструментів у браузері без необхідності

встановлення окремого програмного забезпечення. Робота з API починається з виклику navigator.requestMIDIAccess(), після чого застосунок отримує доступ до об’єктів MIDIInput і MIDIOutput[2]. За допомогою методу send() можна передавати будь-яке стандартне MIDI-повідомлення[2].

Головні функції API, важливі для проекту:

- через властивості inputs та outputs отримати список усіх підключених MIDI-пристроїв.
- метод send() дозволяє передавати дані на вибраний вихідний пристрій, наприклад Note On, Note Off[2]
- кожен вхідний пристрій має подію onmidimessage[2], яка спрацьовує при надходженні MIDI-повідомлення – її можна використовувати для реакції інтерфейсу чи запису нот.
- API дозволяє відстежувати зміну списку пристроїв за допомогою подій statechange[2].
- кожне повідомлення має timestamp, що дозволяє точно синхронізувати час надсилання й обробки подій у реальному часі.

Але серед обмежень які можуть бутим важливими, на які варто звернути увагу воно має:

- API підтримується лише у частині браузерів (наприклад Google Chrome, Microsoft Edge)[5].
- потрібно використовувати HTTPS на сервері для безпечного доступу до пристроїв в мережі.

Web MIDI API підходить для цього проекту, оскільки дозволяє напряму взаємодіяти з MIDI-пристроями прямо з браузера, без потреби в установці додаткового програмного забезпечення. Він забезпечує кросплатформенність, простоту підключення, підтримку реального часу та гнучкість у створенні користувацьких інтерфейсів, що ідеально відповідає цілям розробки доступного та розширюваного MIDI-контролера.

2.4 Апаратне забезпечення для тестування

Для тестування веб-застосунку було обрано Elektron Digitakt(рисунок 2.3) – сучасний багатоканальний пристрій, що об’єднує функції драм-машини, семплера та синтезатору[6]. Цей пристрій підтримує повноцінну роботу з MIDI In/Out, а також має інтерфейс USB MIDI, що дозволяє напряму взаємодіяти з комп’ютером без встановлення додаткових драйверів.

Рисунок 2.3 – Багатоканальний синтезатор Elektron Digitakt

Окрім основних можливостей, важливою перевагою Digitakt є підтримка кількох незалежних MIDI-треків, кожен з яких може працювати на окремому каналі (до 16 одночасно)[6]. Це дає змогу перевіряти:

- одночасну взаємодію з кількома каналами;
- зміну інструментів у режимі реального часу;
- незалежне керування різними параметрами різних треків;
- синхронізацію кількох потоків повідомлень.
- також пристрій може відправляти та приймати повідомлення усіх типів, що дозволяє протестувати повний набір функціоналу застосунку.

Такий вибір апаратного забезпечення гарантує сумісність, стабільність та максимальну відповідність реальним умовам використання.

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

У цьому розділі описується загальна архітектура веб-застосунку для керування MIDI-пристроями.

3.1 Вимоги до функціональності

Вимоги до функціональності застосунку передбачають реалізацію основного набору його можливостей. Застосунок повинен дозволяти вибір MIDI-пристроїв, створення окремих послідовностей, розміщення нот на часовій сітці, керування темпом, каналами та параметрами відтворення, тощо.

3.1.1 Функціональні вимоги

1. застосунок має знаходити всі підключені пристрої(midi outs)
2. застосунок має мати керування початком відтворення музичної послідовності(start)
3. застосунок має мати керування зупинкою відтворення музичної послідовності(stop)
4. повинна бути реалізована функція зміни темпу(tempo) в діапазоні 1-300 ударів на хвилину
5. застосунок має підтримувати до 8 одночасно підключених пристроїв(track)
6. кожен з підключених пристроїв(track) повинен мати окремі налаштування пристрою виводу(midi out)
7. кожен з підключених пристроїв(track) повинен мати окремі налаштування каналу виводу(midi channel), від 1 до 16
8. застосунок повинен мати 16 збережених наборів музичних

послідовностей(patterns)

9. кожен збережений набір послідовностей(pattern) повинен включати усі окремі 8 пристроїв(track)

10. музична послідовність(pattern) повинна мати довжину 16 сегментів розділених на 4 такти темпу

11. повинна бути можливість увімкнути ноту на кожному з 16 сегментів, які входять у послідовність(тобто максимум 16 нот підряд на 1 послідовність)

12. кожна нота повинна мати 3 налаштування: номер ноти(note number C0-C5 - від ноти До 0 октави до ноти До 5 октави), інтенсивність програвання ноти(velocity 0-127), довжина активної фази програшу ноти(length 0-127)

13. інтерфейс повинен мати поле вводу назви проекту

14. застосунок повинен мати завантажування/збереження всього проекту в JSON файл

3.1.2 Нефункціональні вимоги

1. інтерфейс застосунку повинен дозволяти швидко перемикатися між збереженими послідовностями(patterns) в режимі реального часу

2. інтерфейс повинен візуально відображати зміст музичної послідовності та маркер часу, який вказує на поточне положення у послідовності

3. перемикання між інструментами(track) та наборами послідовностей(patterns) не повинно видаляти введену інформацію

3.1.3 Ієрархія даних

Для керування MIDI-пристроями структура проекту організована ієрархічно, щоб підтримувати складну, але гнучку систему музичних послідовностей. Основна логіка побудована навколо 16 послідовностей

(patterns), які об’єднують у собі 8 трекових доріжок пристроїв (tracks) кожен. Кожна з доріжок(track) містить власний набір нот та налаштувань(midi out, midi channel).

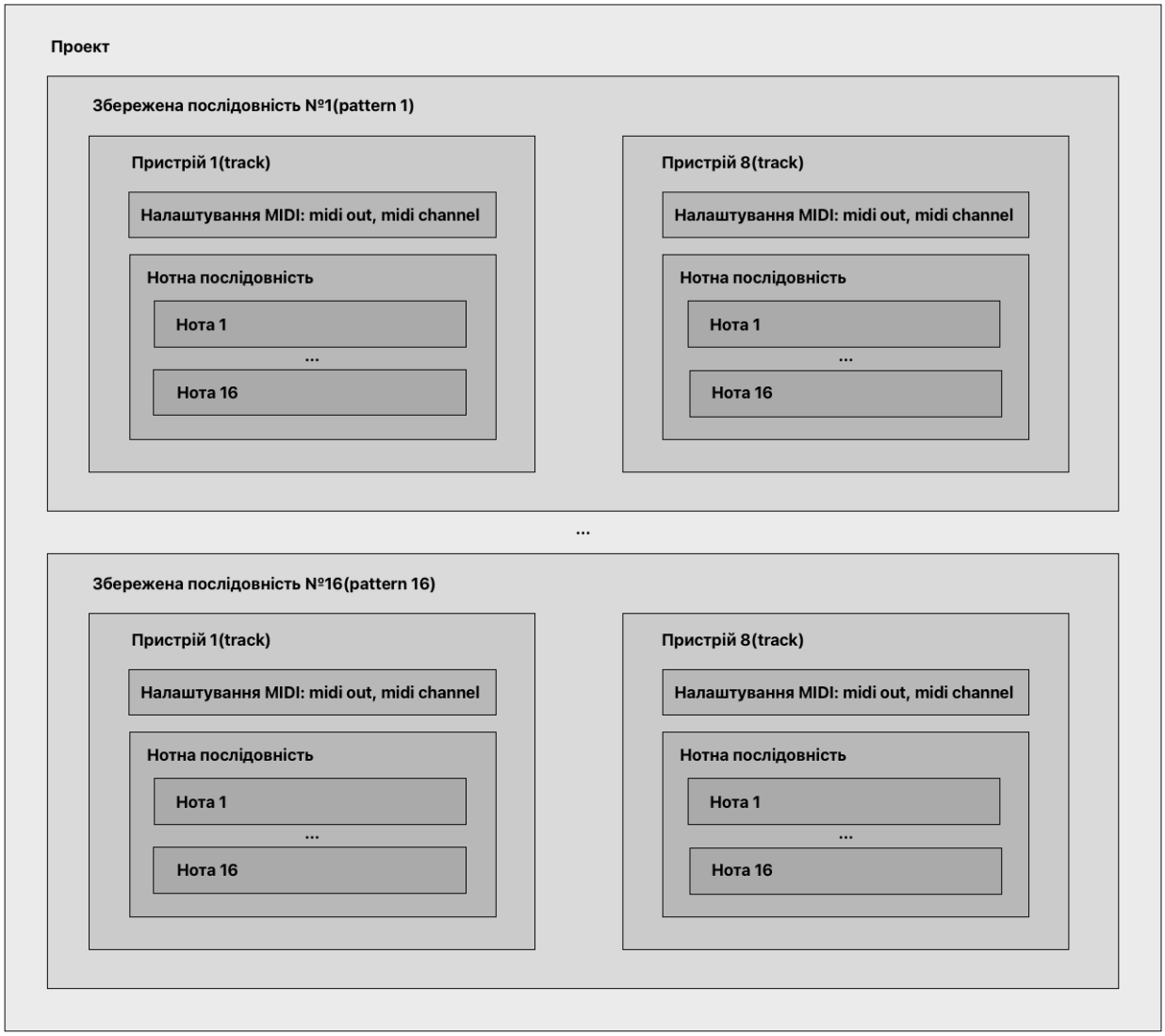

Рисунок 3.1 – Ієрархічна структура даних застосунку

3.2 Інтерфейс користувача

У цьому розділі описано елементи інтерфейсу користувача, що реалізовані у застосунку. Кожен підрозділ відповідає окремій функціональній частині інтерфейсу: від базових кнопок керування до візуалізації нот та редагування їхніх властивостей. Нумерація пунктів у тексті відповідає

позначкам на рисунку макету(рисунок 3.1), що дозволяє легко ідентифікувати розташування кожного елемента в загальній структурі інтерфейсу.

3.2.1 Базові елементи користувача

До базових елементів контролю входять головні налаштування, які визначають загальний стан додатку (програвання та зупинка, збереження та завантаження тощо):

1. Кнопка старту послідовності. При натисканні послідовність починає програватися, а дані надсилаються до підключених пристроїв.
2. Кнопка зупинки послідовності. При натисканні послідовність зупиняється, а загальний стан програвання повертається на початок.
3. Поле введення назви збереженого проєкту. Текст із цього поля буде записано в JSON-файл як назву проекту.
4. Кнопки збереження та завантаження проєкту. Зберігають проєкт на комп’ютер у вигляді JSON-файлу або завантажують проєкт, оновлюючи всю ієрархію даних додатку та його стан.
5. Керування темпом. Числове значення темпу змінюється тягненням курсора вгору або вниз. Темп визначає швидкість програвання послідовності та вимірюється у кількості кроків на хвилину (tempo).

3.2.2 Керування послідовностями

Керування послідовностями забезпечує завантаження збережених послідовностей(pattern) у реальному часі, без зупинки програвання. Також тут реалізовано керування обраною послідовністю для однієї доріжки одночасно:

6. Кнопка вибору збереженої послідовності(pattern). Панель містить 16 таких кнопок – по одній для кожної з 16 послідовностей. При натисканні

змінюється активний набір нот для всіх 8 доріжок(track), що програються одночасно.

7. Кнопка вибору доріжки пристрою. Панель має 8 кнопок – для кожного з 8 пристроїв відповідно. Натискання змінює активну доріжку для редагування у інтерфейсі, не зупиняючи програвання.

8. Керування активним пристроєм(midi out) для обраної доріжки. Обирається тягненням курсора вгору або вниз. У список входять усі доступні в додатку MIDI-пристрої.

9. Вибір MIDI-каналу для обраного пристрою(midi channel). Канал змінюється тягненням курсора вгору або вниз у діапазоні 1–16, згідно зі специфікацією MIDI.

10. Вибір довжини послідовності для обраної доріжки (length). Значення змінюється тягненням курсора в межах 4–16. Число 4 – мінімальна музично осмислена фраза, 16 – максимальна доступна довжина для інтерфейсу.

3.2.3 Візуалізація музичних даних

Це неінтерактивна секція інтерфейсу, яка демонструє активну послідовність нот, їхні властивості та допомагає користувачу орієнтуватися у реальному часі:

11. Візуалізація кожної з 16 нот послідовності. Елемент відображається, якщо нота активна, і зникає, якщо деактивована.

- Висота елементу відповідає висоті ноти: від «До» 0 октави до «До» 5 октави.
- Довжина елементу відповідає тривалості відтворення (від 1/16 до повної довжини секвенції).
- Колір вказує на інтенсивність: чим темніший – тим більша velocity (сила натискання клавіші).
- Під час програвання активна нота підсвічується акцентним

кольором(курсор послідовності).

3.2.4 Керування нотами

Секція керування нотами має чотири панелі: вибір параметру ноти, елементи керування, текстові значення, кнопки активації:

12. Кнопка перемикання до керування висотою ноти (note number). Змінює режим інтерфейсу, не змінюючи значення.

13. Кнопка перемикання до керування інтенсивністю ноти (velocity). Аналогічно змінює режим інтерфейсу, без зміни даних.

14. Кнопка перемикання до керування тривалістю ноти (duration). Перемикає інтерфейс у відповідний режим.

15. 16 елементів зміни значення нот для обраної характеристики. Кожна з 16 нот має свій контролер, значення змінюється тягненням курсора вгору/вниз. Зміни застосовуються в реальному часі і впливають на візуалізацію та поведінку ноти.

16. Текстове відображення обраної характеристики.

- Для ноти(note number): формат нота + октава (наприклад, «C1»).
- Для інтенсивності(velocity): значення у межах 1–127.
- Для тривалості(duration): значення від 1 до 16, що відповідає довжині послідовності.

17. Кнопки активації/деактивації ноти.

- У активному стані нота відтворюється та передається у MIDI.
- У неактивному стані – не грає, не відображається, керування недоступне.

3.2.5 Моніторинг подій

18. Технічні повідомлення про події. Виводяться повідомлення, які оброблені системою. Це корисно для налагодження та аналізу в реальному

часі.

3.2.6 Фінальний макет застосунку

З урахуванням усіх описаних елементів, фінальний макет(рисунок 3.2) має вертикальну побудову, яка відповідає логікі та ієрархії застосунку. Панель моніторингу подій розміщена праворуч, щоб бути завжди в полі зору користувача та ефективно інформувати про поточні дії в системі.

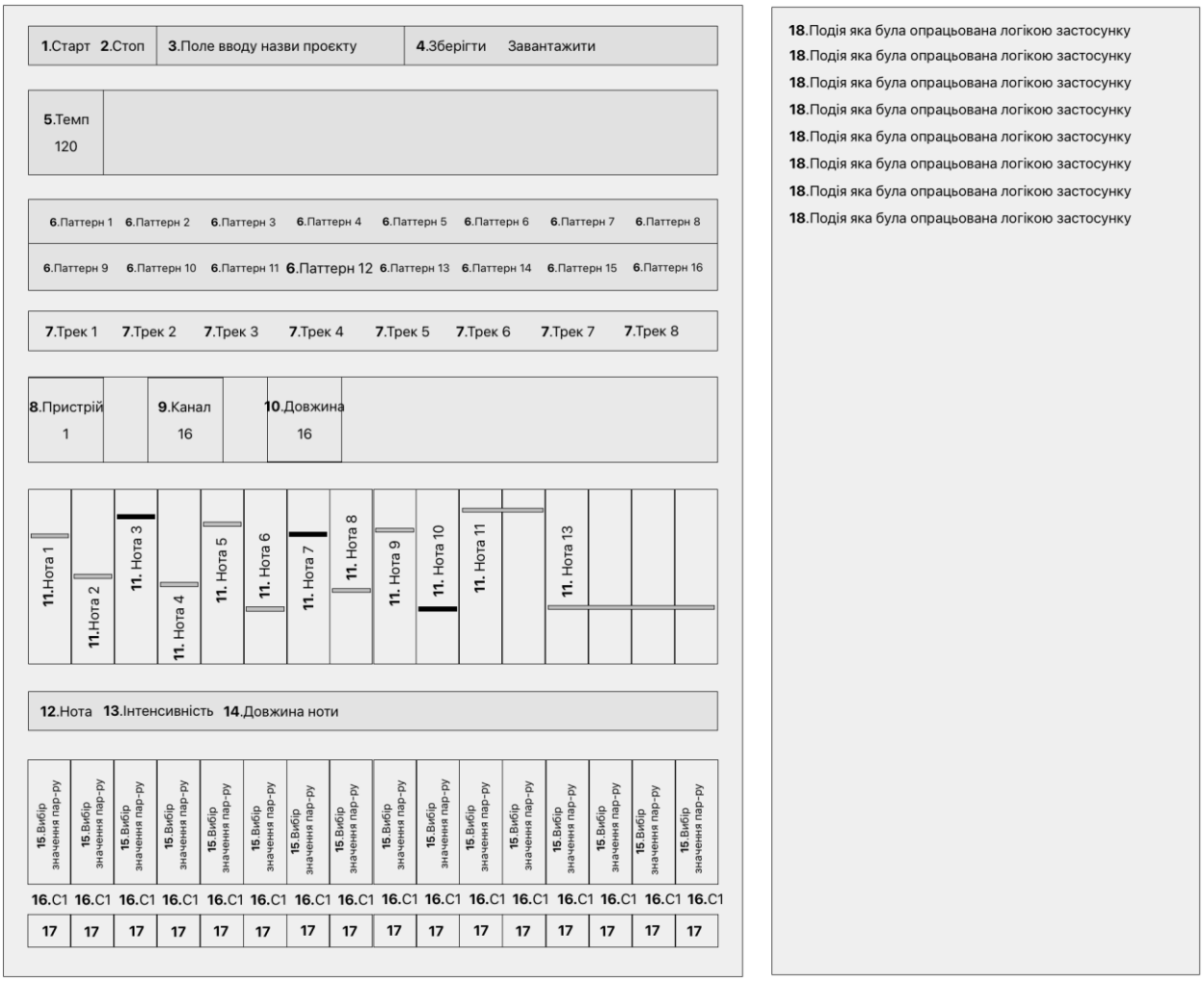

Рисунок 3.2 – Фінальний макет застосунку

3.3 Загальна архітектура застосунку

Загалом, застосунок реалізує класичну компонетно-подієву

архітектуру(Event-driven Modular Architecture)[7], де:

- дані, логіка і візуалізація розділені
- події слугують основою для зв’язку між модулями
- компоненти інкапсульовані та легко масштабуються

Комунікація між модулями відбувається за допомогою системи подій, що дозволяє ефективно організувати реакцію на зміну стану, взаємодію з користувачем і MIDI-пристроями.

Основу архітектури застосунку становлять такі компоненти(рисунок 3.3):

- графічний інтерфейс – керує елементами інтерфейсу користувача та їх логікою (кнопки, панелі, індикатори, відображення нот).

- конфігурації – зберігає всі налаштування та допустимі значення (ноти, темп, довжина патернів, MIDI канали тощо).

- дані – містить поточний стан застосунку, включаючи активний трек, патерн, темп та збережені патерни.

- модуль головної логіки – відповідає за основну логіку: метроном, MIDI-вивід, секвенсор та взаємодію з апаратурою.

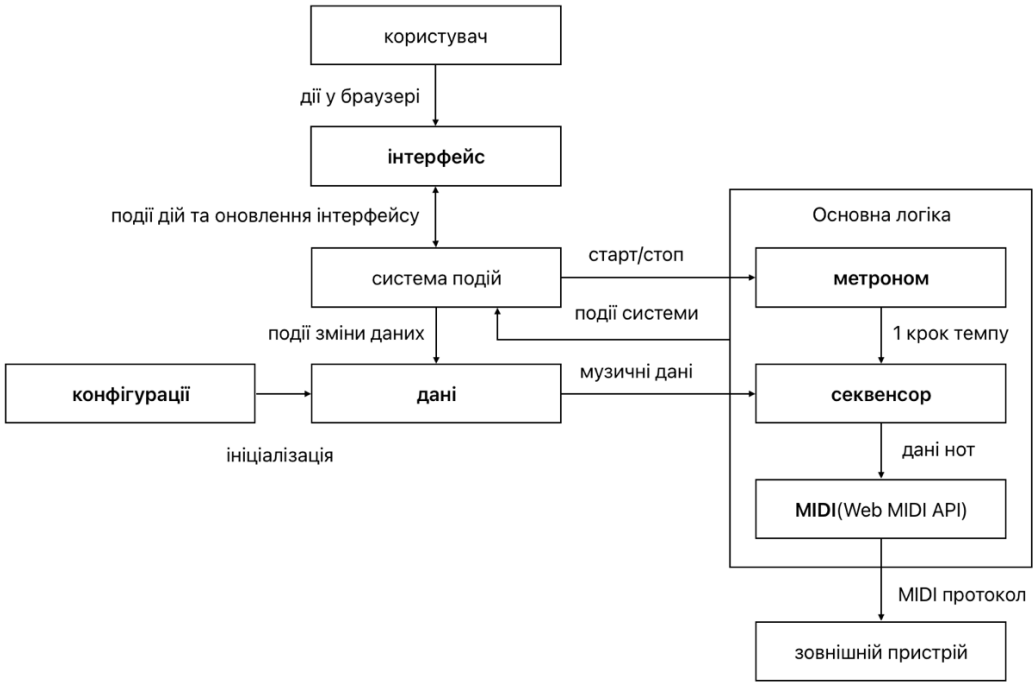

Рисунок 3.3 – Схема базової архіткутури

3.3.1 Конфігураційний модуль

Конфігураційний модуль виконує роль центрального сховища усіх базових та початкових параметрів, які визначають поведінку, логіку та обмеження застосунку. Він забезпечує єдину точку налаштування, де зосереджено всі значення за замовчуванням, допустимі діапазони, а також структури, що регулюють взаємодію користувача з інтерфейсом та MIDI-пристроями. Завдяки цьому модулю інші частини системи можуть працювати узгоджено, покладаючись на передбачувані константи, що спрощує підтримку, розширення та адаптацію проекту під різні сценарії використання.

3.3.2 Модуль даних

У концепції модуль даних виконує роль централізованого сховища всього стану системи – від структур послідовностей до налаштувань користувача. Він забезпечує доступ до поточного патерну, активної доріжки, налаштувань темпу, каналів, пристроїв, параметрів нот і структури композиції загалом.

Цей модуль відповідальний за генерацію стартових даних при ініціалізації, зберігання змін, внесених через інтерфейс, та підтримку внутрішньої логіки переходу між патернами, доріжками й окремими кроками.

Модуль також виступає джерелом для збереження/завантаження проекту у вигляді структури (наприклад, JSON).

3.3.3 Метроном

Метроном розглядається як центральний елемент синхронізації, що забезпечує стабільний часовий поділ для всіх музичних подій у системі. Його основною функцією буде генерація ритмічних «тіків» із заданою точністю

відповідно до темпу, встановленого користувачем.

Особливістю метронома стане використання високоточної часової логіки, яка дозволить попередньо планувати події у часі наперед, мінімізуючи затримки.

Метроном буде інтегрований у єдину систему подій застосунку, транслюючи ритмічні сигнали у вигляді подій для секвенсора, візуального курсору, моніторингу та інших частин інтерфейсу. Завдяки цьому забезпечується єдність часу у всій логікі програвання, включно з точним відтворенням, візуалізацією та взаємодією користувача.

3.3.4 MIDI модуль

MIDI модуль у структурі застосунку слугує абстракцією для взаємодії з апаратними або програмними MIDI пристроями. Його основна роль – надсилання та, потенційно, приймання MIDI повідомлень відповідно до параметрів, визначених у структурі даних (нота, канал, інтенсивність, тривалість тощо) через абстракцію над Web MIDI API.

Цей модуль формує MIDI повідомлення у відповідному форматі та керує маршрутизацією сигналів на вибраний пристрій і канал.

MIDI модуль також відповідальний за отримання доступу до MIDI пристроїв, оновлення списку доступних виходів і налаштування параметрів у реальному часі. Таким чином, він є містком між внутрішньою логікою застосунку та зовнішнім музичним середовищем, забезпечуючи точну, часово чутливу передачу команд без затримок.

3.3.5 Логіка музичних послідовностей(Секвенсор)

Модуль логіки музичних послідовностей(Секвенсор) відповідає за керування кроковими структурами, які визначають, які ноти і з якими параметрами повинні звучати у заданий момент часу. Його головна задача –

синхронізовано з метрономом активувати відповідні кроки у кожному треку та генерувати відповідні музичні події.

Цей модуль зберігає стан крокового лічильника та визначає, який саме крок кожної доріжки є поточним. Якщо на цьому кроці активована нота, модуль передає інформацію про неї до MIDI модуля, включаючи висоту ноти, гучність, довжину, канал і вихідний пристрій. Послідовність може мати різну довжину, що дозволяє створювати складні, асинхронні ритмічні структури між треками.

У майбутньому функціонал цього модуля може бути розширено за рахунок алгоритмічного створення патернів, підтримки варіативності (наприклад, ймовірності для кожної ноти), або імпорту шаблонів із зовнішніх джерел. Таким чином, логіка музичних послідовностей – це центр генерації звуку, яке відповідає за музичну динаміку, структуру та ритм у системі.

3.3.6 Взаємодія з графічним інтерфейсом

Модуль графічного інтерфейсу відповідає за відображення поточного стану системи, надання користувачеві засобів взаємодії та редагування музичних параметрів у візуально зрозумілій формі. Його завдання – синхронізувати графіку з внутрішніми даними додатку, відображати нотні послідовності, параметри треків, MIDI-виходи, а також отримувати і обробляти дії користувача.

Особливістю реалізації інтерфейсу є подієвий підхід: кожна зміна (наприклад, редагування ноти, вибір каналу або запуск програвання) генерує подію, яку система обробляє окремо. Це дозволяє відділити логіку від візуального представлення, підвищуючи масштабованість архітектури. Важливою частиною інтерфейсу є модулі візуалізації нот, що відображають параметри ноти через позицію, колір, висоту і розмір, а також контрольні елементи – слайдери, кнопки, тощо.

3.4 Сценарій використання(Use Case)

Користувачем додатку може бути музикант, розробник, викладач тощо. Основним сценарієм використання буде швидке підключення одного або двох пристроїв в компактних умовах, де необхідне швидке підключення без додаткових налаштувань.

Попередні умови:

- До комп’ютера підключено хоча б один MIDI пристрій.
- Веб-застосунок запущено у сумісному браузері з підтримкою Web MIDI API.
- Доступ до MIDI портів дозволено користувачем

Дії користувача:

1. Користувач відкриває веб-застосунок.
2. Надає дозвіл Web MIDI API на доступ до пристроїв.
3. Вибирає один з доступних MIDI виходів.
4. Обирає MIDI канал для треку.
5. Налаштовує темп.
6. Увімкнувши активні кроки, задає їм ноти, інтенсивність та довжину.
7. Натискає Start – метроном запускає відтворення.
8. Секвенсор програє активні ноти, передаючи їх у MIDI пристрій.
9. Користувач не зупиняючи відтворення змінює параметри в реальному часі.

4 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Підготовка середовища

Перед початком безпосередньої розробки необхідно налаштувати відповідне середовище, яке дозволяє ефективно розробляти, запускати й тестувати застосунок. Це включає встановлення потрібного програмного забезпечення, організацію структури проекту та запуск локального сервера для взаємодії з Web MIDI API.

Для розробки використовується стандартний фронтенд-стек (HTML, CSS, JavaScript), а також ми повинні мати можливість запуску локального сервера. Тож нам необхідно встановити наступне забезпечення:

- node.js – забезпечує запуск локального сервера
- редактор коду – наприклад, Visual Studio Code
- git – для зберігання та контролю версій коду
- Google Chrome – браузер з підтримкою Web MIDI API

Проект не потребує складної інфраструктури. Файлова структура всього проекту має такий вигляд:

Лістинг 4.1 – Файлова структура проекту.

```
/project
  /public
    index.html
    styles.css
    app.js
  /server
    server.js
```

Для відслідковування версій проекту, та безпечної розробки, активуємо git та зробимо перший базовий коміт:

Лістинг 4.2 – Ініціалізація репозиторію проекту.

```
git init
git add .
git commit -m “initial commit: file structure”
```

Оскільки Web MIDI API потребує безпечного контексту (HTTPS або localhost), для тестування в локальному середовищі необхідно підняти простий сервер.

Встановимо пакет для серверу express.js:

Лістинг 4.3 – Встановлення серверного середовища.

```
npm init -y
npm install express
```

Редагуємо файл server.js, додаючи цей код:

Лістинг 4.4 – Код серверу у файлі server.js.

```
const express = require('express');
const app = express();
const PORT = 3000;

app.use(express.static('public'));

app.listen(PORT, () => {
  console.log(`running MIDI apppication on:
http://localhost:${PORT}`);
});
```

Цей код запускає сервер, який буде мати доступ до всіх файлів папки public та подавати їх в браузер до клієнту. Для запуску серверу виконуємо команду node ./server/server.js

У файлі index.html додаємо головний HTML код, який створює основу структури сторінки та контейнер для всіх елементів керування, підключає файли стилів та логіки:

Лістинг 4.5 – Код базової HTML сторінки в файлі index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>MIDI контроллер</title>
  <link rel="stylesheet" href="./styles.css">
</head>
<body>
     <main id="app-container">
          <section id="sequencer-ui">
          </section>
     </main>
     <script src="./app.js"></script>
</body>
</html>
```

Підготуємо стилі в файлі styles.css, які будуть корисні для проекту загалом. Ці стилі відміняють деякі стандартні правила стилізації, які включені в браузер за замовчуванням, та додає змінні кольорів та розмірів, які ми будемо використовувати в проекті:

Лістинг 4.6 – Код базових стилів в файлі styles.css.

```
:root {
  --background-color: #000000;
  --gap: 20px;
  --half-gap: 10px;
  --double-gap: 40px;
  --green-color: #04ff3af1;
  --radius: 8px;
  --inner-radius: 4px;

  --main-color: #d8bcc9;
  --neutral-color: #645159;
  --background-light-color: #292929;

}

button {
  all: unset;
  cursor: pointer;
}

* {
  box-sizing: border-box;
  user-select: none;
}

input, textarea {
  user-select: text;
```

```
}

input,
textarea,
select,
button {
  margin: 0;
  padding: 0;
  border: none;
  background: none;
  font: inherit;
  color: inherit;
  box-sizing: border-box;
  outline: none;
}

input::placeholder {
  color: var(--background-light-color);
}

#app-container {
  display: flex;
  width: 800px;
  margin: var(--gap) auto;
  margin-top: var(--double-gap);
  border-radius: var(--radius);
}
```

Для початку роботи з JavaScript додаємо необхідний код в app.js. Цей код важливий, так як він прослуховує подію остаточного завантаження документу, щоб впевнитись, що всі елементи повністю доступні до взаємодії з ними. Додатково, створимо головний об’єкт App, що і буде нашим додатком, керуючі елементами, взаємодією між компонентами а також має зберігати усі дані.

Лістинг 4.7 – Код базового об’єкту логіки застосунку у файлі app.js.

```
document.addEventListener('DOMContentLoaded', function () {
     const App = {
          config: {},
          data: {},
          UI: {},
          Engine: {},
     }
}
```

4.2 Реалізація користувацького інтерфейсу

Реалізовувати інтерфейс будемо послідовно - від головних елементів до деталей. Починаючи з секцій та розділення сторінки, закінчуючи елементами контролю та прив’язкою їх до логіки застосунку. Тож, спочатку ми додаємо розмітку та стилізовану структуру, а потім ми напишемо JavaScript код, який буде відслідковувати дії користувача та форматувати дані для обробки логікою, тому кожен розділ буде складатися з 3 кроків: HTML розмітка, CSS стилі та JavaScript код.

4.2.1 Головні елементи керування

Згідно з нашим макетом(рисунок 3.2), головними елементами керування є:

- кнопки старт(елемент 1) та стоп(елемент 2)
- поле назви проекту(елемент 3)
- кнопки зберігання та завантаження(елемент 4)
- елемент обрання темпу послідовності(елемент 5)
- 16 кнопок обрання збережених послідовностей(patterns)(елемент 6)
- 8 кнопок обрання доріжки пристрою(track)(елемент 7)

Додаємо код HTML структури, включно з переліченими елементами(код скорочено, без повторень однакових елементів). Цей код ділить структуру на секції головного керування, виклику послідовностей та доріжок, враховуючи ієрархію макету та кількості елементів на ньому.

Лістинг 4.8 – Код головних секцій структури HTML документу.

```
<section id="sequencer-ui">
  <section id="main-controls">
    <button id="start-button" name="start"></button>
    <button id="stop-button" name="stop"></button>
    <input id="project-name" type="text"
```

```
placeholder="project name">
        <button id="save-button" name="save">save</button>
        <button id="load-button" name="load">load</button>
      </section>

      <section id="pattern-controls">
        <div class="control">tempo <span id="tempo-control"
class="selected">120</span></div>
      </section>

      <section id="patterns">
        <div class="pattern selected">
          <div class="pattern-progress">
          </div>
          <div class="pattern-id">ptn0</div>
        </div>
        …
        <div class="pattern">
          <div class="pattern-progress">
          </div>
          <div class="pattern-id">ptn15</div>
        </div>
      </section>

      <section id="track-selection">
        <div class="track selected">
          <div class="selected-indicator"></div>
          <div class="track-id">tr0</div>
        </div>
         …
        <div class="track">
          <div class="selected-indicator"></div>
          <div class="track-id">tr7</div>
        </div>
      </section>
```

Тепер додаємо стилі для головних елементів керування. Для позиціювання та побудови сітки елементів, в цілому було використано стандарти Flexbox та Grid[8].

Лістинг 4.9 – Стилі головної розмітки документу.

```
  #sequencer-ui {
    flex: 1;

    section {
      margin: var(--gap) 0;
    }
```

```
#main-controls {
  margin-top: 0;
  display: flex;
  align-items: center;

  button {
    height: 40px;
    width: 40px;
  }

  #start-button {
    width: 0;
    height: 0;
    border-top: 8px solid transparent;
    border-bottom: 8px solid transparent;
    border-left: 14px solid var(--neutral-color);
    padding: 0;
    border-radius: 0;
    margin-right: var(--gap);

    &:hover, &:active {
      border-left-color: var(--main-color);
    }
  }

  #stop-button {
    width: 14px;
    height: 14px;
    display: block;
    background-color: var(--neutral-color);
    margin-right: var(--gap);

    &:hover, &:active {
      background-color: var(--main-color);
    }
  }

  #project-name {
    background-color: var(--neutral-color);
    padding: 2px;
    color: var(--background-color);
    margin-right: var(--gap);
  }

  #save-button, #load-button {
    color: var(--neutral-color);
    margin-right: var(--gap);
    padding: 2px;

    &:hover, &:active {
      color: var(--main-color);
    }
  }
```

```
}

#patterns {
  display: grid;
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(8, 1fr);
  gap: var(--half-gap);

  .pattern {
    color: var(--background-color);
    display: flex;
    position: relative;
    height: 40px;
    padding: 3px;
    border: solid 2px var(--neutral-color);
    border-radius: var(--radius);
    cursor: pointer;

    &.selected {
      border-color: var(--main-color);
      .pattern-progress {
        background-color: var(--main-color);
      }
    }

    .pattern-progress {
      width: 100%;
      height: 100%;
      border-radius: var(--inner-radius);
      background-color: var(--neutral-color);
    }

    .pattern-id {
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
    }
  }
}

#pattern-controls {
  color: var(--neutral-color);

  .selected {
    color: var(--main-color);
  }
}

#track-selection {
  display: grid;
  grid-template-rows: repeat(1, 1fr);
  grid-template-columns: repeat(8, 1fr);
```

```
    gap: var(--half-gap);

    .track {
      color: var(--background-color);
      display: flex;
      position: relative;
      height: 30px;
      border: solid 2px var(--neutral-color);
      border-radius: var(--inner-radius);
      cursor: pointer;

      &.selected {
        border-color: var(--main-color);

        .selected-indicator {
          background-color: var(--main-color);
        }
      }

      .selected-indicator {
        width: 100%;
        height: 100%;
        background-color: var(--neutral-color);
      }

      .track-id {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
      }
    }
  }
```

За контроль інтерфейсу буде відповідати модуль UI нашого об’єкту App. Цей код пов’язується з елементами сторінки, та створюється, як тільки сторінка буде завантажена. В цілому, код елементів контролю виконує такі дії: змінює тексти та стилі відносно стану об’єкту, створює функції-обробники дій користувача та ініціює системні події, які наш застосунок може відловлювати та робити необхідні зміни в даних. Логіка кнопок та інших елементів контролю, в цілому, не визначає стилі або форму елементів, а тільки редагує набір класів цих елементів, що описані у файлі стилів. Клас ScrollValueControl – відповідає за елементи зміни параметрів та обробки відповідних вхідних значень, а клас ButtonControl – за кнопки та їх стан.

Лістинг 4.10 – JavaScript код логіки елементів контролю.

```
      ScrollValueControl: class {
        constructor(element, scrollId, values, onChange = null,
defaultValue = null) {
          this.el = element;
          this.scrollId = scrollId;
          this.values = values;
          this.onChange = onChange;

          if (defaultValue !== null &&
values.includes(defaultValue)) {
            this.index = values.indexOf(defaultValue);
          } else {
            const currentText = element.textContent.trim();
            this.index = values.indexOf(currentText);
            if (this.index === -1) this.index = 0;
          }

          this._bindEvents();
          this.updateDisplay();
        }

        _bindEvents() {
          let startY = 0;

          const onMouseMove = (e) => {
            const deltaY = e.clientY - startY;
            const steps = Math.round(deltaY / 10);
            if (steps !== 0) {
              this.scroll(-steps);
              startY = e.clientY;
            }
          };

          const onMouseUp = () => {
            document.removeEventListener('mousemove',
onMouseMove);
            document.removeEventListener('mouseup', onMouseUp);
            this.el.style.cursor = 'default';
          };

          this.el.addEventListener('mousedown', (e) => {
            e.preventDefault();
            startY = e.clientY;
            this.el.style.cursor = 'ns-resize';

            document.addEventListener('mousemove', onMouseMove);
            document.addEventListener('mouseup', onMouseUp);
          });
        }

        scroll(delta) {
```

```
          const newIndex = Math.max(0, Math.min(this.index +
delta, this.values.length - 1));
          if (newIndex !== this.index) {
            this.index = newIndex;
            this.updateDisplay();

            if (typeof this.onChange === 'function') {
              this.onChange(this.values[this.index], this.el);
            }

            document.dispatchEvent(new
CustomEvent(`app.ui.${this.scrollId}_scroll.scroll`, {
              detail: { value: this.value }
            }));
          }
        }

        updateDisplay() {
          this.el.textContent = this.values[this.index] ||
'null';
        }

        get value() {
          return this.values[this.index];
        }

        set value(val) {
          const i = this.values.indexOf(val);
          if (i !== -1) {
            this.index = i;
            this.updateDisplay();
          }
        }
      },

      ButtonControl: class {
        constructor(el, buttonId, onChange = null, initialValue
= false) {
          this.el = el;
          this.buttonId = buttonId;
          this.onChange = onChange;
          this.value = initialValue;
          this.updateVisual();
          this._bindEvents();
        }

        _bindEvents() {
          this.el.addEventListener('click', () => {
            this.value = !this.value;
            this.updateVisual();

            if (typeof this.onChange === 'function') {
              this.onChange(this.value);
```

```
            }

            document.dispatchEvent(new
CustomEvent(`app.ui.${this.buttonId}_button.click`, {
              detail: { value: this.value }
            }));
          });
        }

        updateVisual() {
          this.el.classList.toggle('active', this.value);
        }

        setValue(newValue) {
          this.value = !!newValue;
          this.updateVisual();
          if (typeof this.onChange === 'function') {
            this.onChange(this.value);
          }
        }

        getValue() {
          return this.value;
        }
      },
```

Реєструємо ці елементи в системі при завантаженні сторінки наступним кодом:

Лістинг 4.11 – JavaScript код логіки елементів контролю.

```
      this.UI.startButton = new
this.UI.ButtonControl(document.getElementById('start-button'),
'start')
      this.UI.stopButton = new
this.UI.ButtonControl(document.getElementById('stop-button'),
'stop')

      // tempo
      this.UI.tempoControl = new this.UI.ScrollValueControl(
        document.getElementById('tempo-control'),
        'tempo',
        this.config.tempoRange,
        null,
        this.data.tempo
      );

      // pattern selection
      this.UI.patternSelection = new this.UI.ControlSelection(
```

```
    document.querySelector('#patterns'),
    null,
    'pattern',
    '.pattern'
  );

  // track selection
  this.UI.trackSelection = new this.UI.ControlSelection(
    document.getElementById('track-selection'),
    null,
    'track',
    '.track'
  );
```

Як результат маємо першу частину інтерфейсу застосунку, який містить усі головні елементи контролю – дає основу для відтворення композиції, обранням варіацій та налаштуваннями окремих пристроїв, згідно до макету.

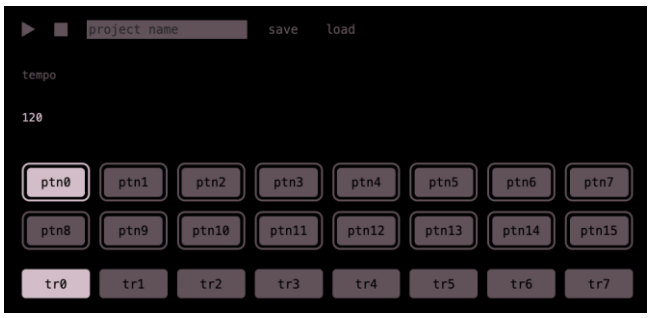

Рисунок 4.1 – Розроблений інтерфейс секції головного керування.

4.2.2 Побудова послідовностей, доріжки

Згідно з макетом, ці секції мають такі елементи:

- налаштування доріжки: вибір пристрою(елемент 8), MIDI каналу(елемент 9), та довжини(елемент 10)
- вибір типу характеристики для контролю значень: номер ноти(елемент 12), інтенсивність(елемент 13), тривалість(елемент 14)
- головні елементи контролю значень - по 1 на кожну ноту(елемент 15)
- індикатор значення, по 1 на кожну ноту(елемент 16)
- кнопки активації ноти, по 1 на кожну ноту послідовності(елемент 17)

Спочатку додаємо HTML код структури цих елементів, відповідно до ієрархії макету(код зменшено, без повторень однакових елементів):

Лістинг 4.12 – HTML розмітка елементів музичних послідовностей.

```
<section id="step-control-select">
   <div class="control selected">note</div>
   <div class="control">velocity</div>
   <div class="control">length</div>
 </section>

 <section id="step-values">
   <div class="step-value">
     <div class="value-indicator"></div>
     <div class="value-id">C1</div>
   </div>
  …
   <div class="step-value">
     <div class="value-indicator"></div>
     <div class="value-id">C1</div>
   </div>
 </section>

 <section id="step-toggles">
   <div class="step-toggle">
     <div class="toggle-indicator"></div>
     <div class="step-id">0</div>
   </div>
   …
   <div class="step-toggle">
     <div class="toggle-indicator"></div>
     <div class="step-id">15</div>
   </div>
 </section>
```

Додаємо стилі, які не входять до частини вже доданого коду, бо деякі елементи мають сумісні характеристики. Важливо підкреслити, що в стилях ми одразу реалізуємо зміни елементів, відповідно до їх стану(такі як стани активації кнопки, стан відключених елементів контролю, тощо). Знов для організації сітки використовуємо Flexbox та Grid[8]:

Лістинг 4.13 – Стилі елементів керування нот.

```
#step-control-select {
  display: flex;
  gap: var(--gap);
```

```
  color: var(--neutral-color);

  .control {
    cursor: pointer;
  }

  .selected {
    color: var(--main-color);
  }
}

#step-values {
  display: grid;
  grid-template-rows: repeat(1, 1fr);
  grid-template-columns: repeat(16, 1fr);
  gap: calc(var(--gap)/4);
  margin-bottom: 40px;
  font-size: 10px;

  .step-value {
    color: var(--neutral-color);
    display: flex;
    position: relative;
    height: 100px;
    padding: 3px;
    border: solid 2px var(--neutral-color);
    border-radius: var(--radius);
    cursor: pointer;
    align-items: flex-end;

    &.active {
      border-color: var(--main-color);
      color: var(--main-color);

      .value-indicator {
        background-color: var(--main-color);
      }
    }

    .value-indicator {
      width: 100%;
      height: 80%;
      border-radius: var(--inner-radius);
      background-color: var(--neutral-color);
    }

    .value-id {
      position: absolute;
      top: 125%;
      left: 50%;
      transform: translate(-50%, -50%);
    }
  }
```

```
}

#step-toggles {
  margin-top: var(--half-gap);
  display: grid;
  grid-template-rows: repeat(1, 1fr);
  grid-template-columns: repeat(16, 1fr);
  gap: calc(var(--gap)/3);

  .step-toggle {
    color: var(--neutral-color);
    display: flex;
    position: relative;
    height: 30px;
    padding: 3px;
    border: solid 2px var(--neutral-color);
    border-radius: var(--radius);
    cursor: pointer;
    align-items: flex-end;

    &.active {
      border-color: var(--main-color);
      color: var(--background-color);

      .toggle-indicator {
        background-color: var(--main-color);
      }
    }

    .toggle-indicator {
      width: 100%;
      height: 100%;
      border-radius: var(--inner-radius);
    }

    .step-id {
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
    }
  }
}
```

JavaScript код логіки цих елементів, частково сумісний з тим що ми вже реалізували, але характер даних, які відображені на цих елементах змушує нас створити окрему реалізацію, для підтримки більш комплексних подій. Код класів цих елементів, в цілому, реалізує наступну логіку: прослуховування дій курсору користувача та доступ до обраних значень,

зміна класів та стилів елементів відповідно до стану даних, запуск відповідних подій в системі, для подальшої їх обробки. Клас StepToggleControl відповідає за кнопки активації нот, ControlSelection за кнопки обрання характеристики, що редагується та ScrollValueControl за значення характеристики для кожної ноти. За налаштування доріжки(midi out, midi channel, length) будуть відповідати вже створені класи ScrollValueControl.

Лістинг 4.14 – JavaScript код логіки елементів управління нотами.

```
      StepToggleControl: class {
        constructor(element, step, onToggle = null) {
          this.el = element;
          this.step = step;
          this.isActive = false;
          this.onToggle = onToggle;
          this._bindEvents();
        }

        toggleActive(isActive) {
          this.isActive = isActive
          this.el.classList.toggle('active', isActive);
        }

        _bindEvents() {
          this.el.addEventListener('click', () => {
            // this.isActive = !this.isActive;
            // this.el.classList.toggle('active');
            this.toggleActive(!this.isActive);

            const status = this.isActive ? 'activated' :
'disabled'

            document.dispatchEvent(new
CustomEvent(`app.ui.step.${status}`, {
              detail: { step: this.step }
            }));

            if (typeof this.onToggle === 'function') {
              this.onToggle(this.isActive)
            }
          });
        }
      },

      ControlSelection: class {
        constructor(el, onSelect = null, selectionId, selector =
```

```
'.control') {
          this.el = el;
          this.controls =
Array.from(el.querySelectorAll(selector));
          this.selectionId = selectionId;
          this.onSelect = onSelect;

          this._bindEvents();
        }

        _bindEvents() {
          this.controls.forEach(control => {
            control.addEventListener('click', () => {
              this.select(control);

              document.dispatchEvent(new
CustomEvent(`app.ui.${this.selectionId}.selected`, {
                detail: { selected: this.selected,
selectedIndex: this.selectedIndex }
              }));
            });
          });
        }

        select(selectedControl) {
          this.controls.forEach(control => {
            control.classList.remove('selected');
          });

          selectedControl.classList.add('selected');

          if (typeof this.onSelect === 'function') {
            this.onSelect(selectedControl);
          }
        }

        selectByIndex(index) {
          if (this.controls[index]) {
            this.select(this.controls[index]);
          }
        }

        selectByElement(el) {
          if (this.controls.includes(el)) {
            this.select(el);
          }
        }

        get selected() {
          return this.controls.find(el =>
el.classList.contains('selected')).textContent.trim() || 'null';
        }
```

```
        get selectedIndex() {
          return this.controls.findIndex(el =>
el.classList.contains('selected'));
        }
      },

      ScrollValueControl: class {
        constructor(element, scrollId, values, onChange = null,
defaultValue = null) {
          this.el = element;
          this.scrollId = scrollId;
          this.values = values;
          this.onChange = onChange;

          if (defaultValue !== null &&
values.includes(defaultValue)) {
            this.index = values.indexOf(defaultValue);
          } else {
            const currentText = element.textContent.trim();
            this.index = values.indexOf(currentText);
            if (this.index === -1) this.index = 0;
          }

          this._bindEvents();
          this.updateDisplay();
        }

        _bindEvents() {
          let startY = 0;

          const onMouseMove = (e) => {
            const deltaY = e.clientY - startY;
            const steps = Math.round(deltaY / 10);
            if (steps !== 0) {
              this.scroll(-steps);
              startY = e.clientY;
            }
          };

          const onMouseUp = () => {
            document.removeEventListener('mousemove',
onMouseMove);
            document.removeEventListener('mouseup', onMouseUp);
            this.el.style.cursor = 'default';
          };

          this.el.addEventListener('mousedown', (e) => {
            e.preventDefault();
            startY = e.clientY;
            this.el.style.cursor = 'ns-resize';

            document.addEventListener('mousemove', onMouseMove);
            document.addEventListener('mouseup', onMouseUp);
```

```
        });
      }

      scroll(delta) {
        const newIndex = Math.max(0, Math.min(this.index +
delta, this.values.length - 1));
        if (newIndex !== this.index) {
          this.index = newIndex;
          this.updateDisplay();

          if (typeof this.onChange === 'function') {
            this.onChange(this.values[this.index], this.el);
          }

          document.dispatchEvent(new
CustomEvent(`app.ui.${this.scrollId}_scroll.scroll`, {
            detail: { value: this.value }
          }));
        }
      }

      updateDisplay() {
        this.el.textContent = this.values[this.index] ||
'null';
      }

      get value() {
        return this.values[this.index];
      }

      set value(val) {
        const i = this.values.indexOf(val);
        if (i !== -1) {
          this.index = i;
          this.updateDisplay();
        }
      }
    },
```

Як і в попередньому випадку, ми реєструємо класи на відповідних елементах реалізованих в HTML:

Лістинг 4.15 – Ініціалізація об’єктів керування нотами.

```
    // midi out
    this.UI.trackMidiOutControl = new
this.UI.ScrollValueControl(
      document.getElementById('midi-out-control'),
      'track.midi_out',
      this.Engine.adapter.outputNames(),
```

```
        null,
this.Engine.adapter.outputNames(this.currentTrack().midiOut)
      );

      // midi ch
      this.UI.trackMidiChannelControl = new
this.UI.ScrollValueControl(
        document.getElementById('midi-ch-control'),
        'track.midi_ch',
        this.config.midiChannels,
        null,
        this.currentTrack().midiChannel
      );

      // pattern length
      this.UI.trackPatternLengthControl = new
this.UI.ScrollValueControl(
        document.getElementById('pattern-length-control'),
        'track.length',
        this.config.patternLengths,
        null,
        this.currentTrack().length
      );

      // step value select
      this.UI.stepControlsSelection = new
this.UI.ControlSelection(
        document.getElementById('step-control-select'),
        null,
        'step_value_control',
        '.control'
      );

      // step toggles
      document.querySelectorAll('.step-toggle').forEach((el, i)
=> {
        this.UI.stepToggles.push(new
this.UI.StepToggleControl(el, i));
      });
```

Як результат отримаємо вже більш комплексний інтерфейс та логіку взаємодії з елементами, відповідно до макету:

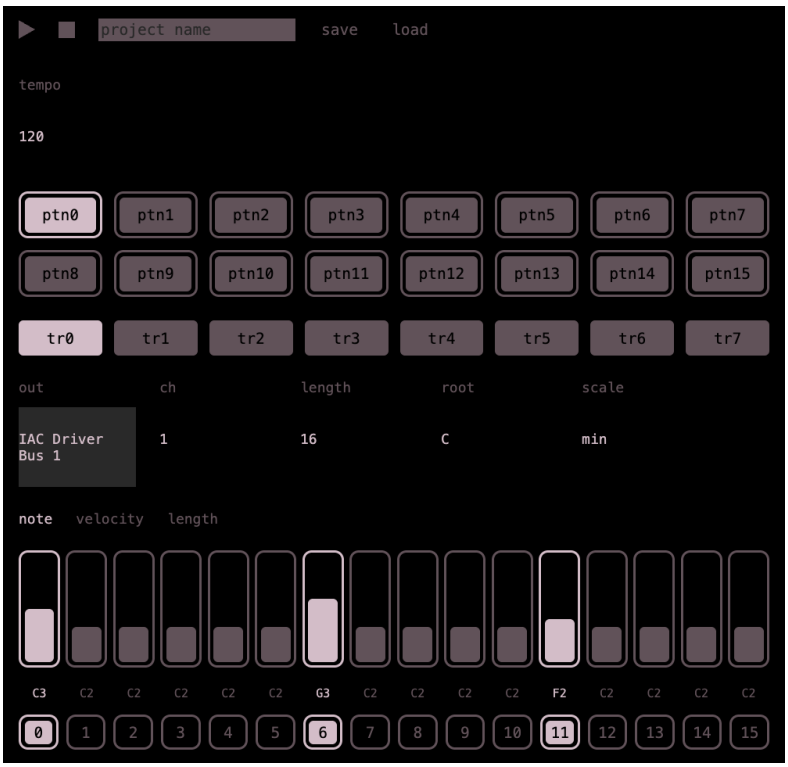

Рисунок 4.2 – Реалізований інтерфейс головних елементів контролю послідовностями нот

4.2.3 Візуалізація послідовності

Для більш ефективної взаємодії з інтерфейсом, макет додатку містить візуалізацію послідовності(елемент 11), де представлено 16 секцій послідовності, а ноти та їх характеристики розміщенні по одній на секцію. Також нам потрібен курсор, як елемент візуалізації покрокового прогресу послідовності в часі.

HTML код цієї секції, це 16 однакових елементів, тож код спрощено, без повторень.

Лістинг 4.16 – Розмітка візуалізації нот в послідовності.

```
<section id="sequence-overview">
  <div class="step active">
    <div class="note"></div>
  </div>
  …
```

```
    <div class="step">
      <div class="note"></div>
    </div>
  </section>
```

Код стилів для цієї секції не використовує елементи які ми вже реалізували, і має свої особливі функції. У кожній із 16 секцій знаходиться елемент ноти, висота звуку якої буде характеризована відстанню від нижнього краю всієї секції(технічно це буде реалізовано CSS правилами bottom). Прозорість елемента ноти(CSS правило opacity) буде характеризувати значення інтенсивності ноти(velocity), а тривалість(duration) характеризуватиме довжина елементу(width в процентних значеннях). Якщо нота буде вимкнута, то її прозорість буде максимальною, тобто елемент візуально не помітний. Окремою функцією секції є курсор покрокової прогресії послідовності, який буде реалізовано зміною кольору фону сегменту, відповідного до часу програвання послідовності.

Лістинг 4.17 – Стилі візуалізації послідовності нот та курсора прогресії.

```
#sequence-overview {
  height: 140px;
  border-radius: var(--radius);
  background-color: var(--background-light-color);
  display: grid;
  grid-template-rows: repeat(1, 1fr);
  grid-template-columns: repeat(16, 1fr);
  overflow: hidden;

  .step {
    height: 100%;
    display: flex;
    align-items: flex-end;
    position: relative;
    z-index: 1;

    &:not(:last-child) {
      border-right: 1px dashed var(--background-color);
    }

    &:last-child {
      border: none;
    }

    &:nth-child(4n) {
```

```
        border-right: 1px solid var(--background-color);
      }

      &.active {
        background-color: var(--neutral-color);
      }

      .note {
        position: absolute;
        width: 100%;
        height: 0;
        border-top: 4px solid var(--main-color);
        opacity: .5;
        z-index: 2;
      }
    }
  }
```

Логіка візуалізації послідовності нот зав’язана на значеннях характеристик цих нот, обраних користувачем. Тому клас, який відповідає за цю секцію, повинен мати методи контролю відображення відповідних елементів нот, руху курсору відтворення послідовності, відповідно до вхідних даних інтерфейсу. Варто відмітити функцію advanceCursor(), яка отримавши поточний набір нот та їх характеристик, в циклі обирає ноту що повинна бути відтворення у даний момент та змінює для неї стилі, реалізуючи курсор прогресії послідовності. Таким чином користувач має візуальну орієнтацію на якій стадії відтворення знаходиться послідовність.

Лістинг 4.18 – Логіка візуалізації послідовності нот та курсора прогресії.

```
    SequenceOverview: class {
      constructor(el, app = {}) {
        this.el = el;
        this.noteValues = app.config.notes;
        this.patternLength = app.currentTrack().length;
        this.app = app;
        this.noteElements =
Array.from(el.querySelectorAll('.note'));
        this.stepElements =
Array.from(el.querySelectorAll('.step'));
        this.notes = [];
        this.cursorPosition = 0;
      }

      updateNotes(notes) {
```

```
        this.notes = notes;
        this.render();
      }

      render() {
        this.noteElements.forEach((el, i) => {
          const note = this.notes[i];

          if ((i+1) > this.app.currentTrack().length) {
            this.stepElements[i].style.display = 'none'
          } else {
            this.stepElements[i].style.display = 'flex'
          }

          if (!note.active) {
            el.style.opacity = '0';
            el.style.width = '';
            el.style.bottom = '';
            el.textContent = '';
            return;
          }

          el.style.opacity =
this.mapVelocityToOpacity(note.velocity);
          el.style.width = this.mapLengthToWidth(note.length);
          el.style.bottom = this.mapNoteToHeight(note.note);
        });
      }

      mapVelocityToOpacity(velocity) {
        const min = 0.1, max = 1;
        const clamped = Math.max(0, Math.min(127, velocity));
        return ((clamped / 127) * (max - min) +
min).toFixed(2);
      }

      mapLengthToWidth(length) {
        const clamped = Math.max(1, Math.min(16, length));
        return `${clamped * 100}%`;
      }

      mapNoteToHeight(noteName) {
        const index = this.noteValues.indexOf(noteName);
        const maxIndex = this.noteValues.length - 1;

        if (index === -1 || maxIndex === 0) return '0%';
        return `${(index / maxIndex) * 100}%`;
      }

      advanceCursor() {
        this.stepElements.forEach((el, i) => {
          el.classList.remove('active');
        })
```

```
          if ((this.cursorPosition + 1) ==
this.app.currentTrack().length) {
            this.cursorPosition = 0
          } else {
            this.cursorPosition += 1
          }


this.stepElements[this.cursorPosition].classList.add('active')
        }

        resetCursor() {
          this.cursorPosition = 0
          this.stepElements.forEach((el, i) => {
            el.classList.remove('active');
          })

          this.stepElements[0].classList.add('active')
        }
      },
```

Лістинг 4.19 – Ініціалізації об’єкту класу логіки візуалізації послідовності

```
this.sequenceOverview ||= new this.SequenceOverview(
  document.getElementById('sequence-overview'),
    App
  )
this.sequenceOverview.updateNotes(App.currentTrack().steps)
```

Як результат, маємо повністю робочу секцію візуалізації музичних даних, відповідно до макету.

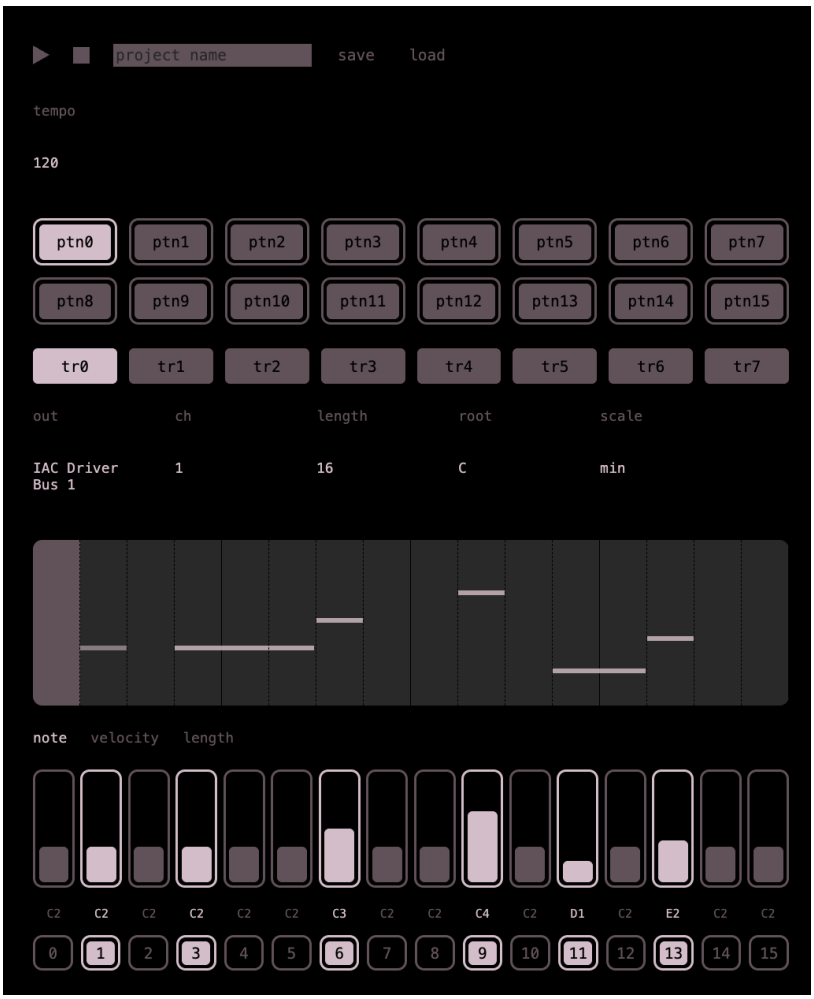

Рисунок 4.3 – Реалізований інтерфейс усіх елементів керуваня

4.2.4 Моніторинг подій

Останнім важливим елементом макету є список останніх подій, які були опрацьовані в системі(елемент 18). Ця інформація важлива для розуміння як працює система, які дії впливають на стан даних. Це особливо корисно на етапі розробки логіки застосунку, тому цей елемент розташовано окремо з правого краю сторінки.

Лістинг 4.20 – Розмітка списку моніторингу подій системи

```
<aside id="monitoring-log">
  <div class="log-item">application init</div>
</aside>
```

Лістинг 4.21 – Стилі списку моніторингу подій системи

```
#monitoring-log {
  width: 160px;
  font-size: 5px;
  padding-left: var(--gap);

  .log-item {
    margin-bottom: 5px;
  }
}
```

Характерною рисою цього елементу є інтенсивне наповнення рядками подій, які трапляються в системі дуже часто. Тому варто додати функцію самоочищення та ліміту кількості рядків(обрано 48 рядків).

Лістинг 4.22 – Логіка списку моніторингу подій системи

```
MonitoringLog: class {
  constructor(el) {
    this.el = el;
    this.maxItems = 48;
  }

  log(e, obj) {
    const logItem = document.createElement('div');
    logItem.className = 'log-item';
    logItem.textContent = `${e}: ${JSON.stringify(obj)}`;

    this.el.appendChild(logItem);

    while (this.el.children.length > this.maxItems) {
      this.el.removeChild(this.el.firstChild);
    }
  }
},
```

Лістинг 4.23 – Ініціалізація об’єкту моніторингу подій системи

```
// moitoring log
this.UI.monitoringLog = new this.UI.MonitoringLog(
  document.getElementById('monitoring-log')
)
```

Тепер наш інтерфейс виглядає більш повноцінно і повністю охоплює деталі розробленого макету. Додатково, кожен з елементів інтерфейсу має набор усіх необхідних функцій для отримання даних взаємодії користувача, зміни стану інтерфейсу та реєстрації подій в системі.

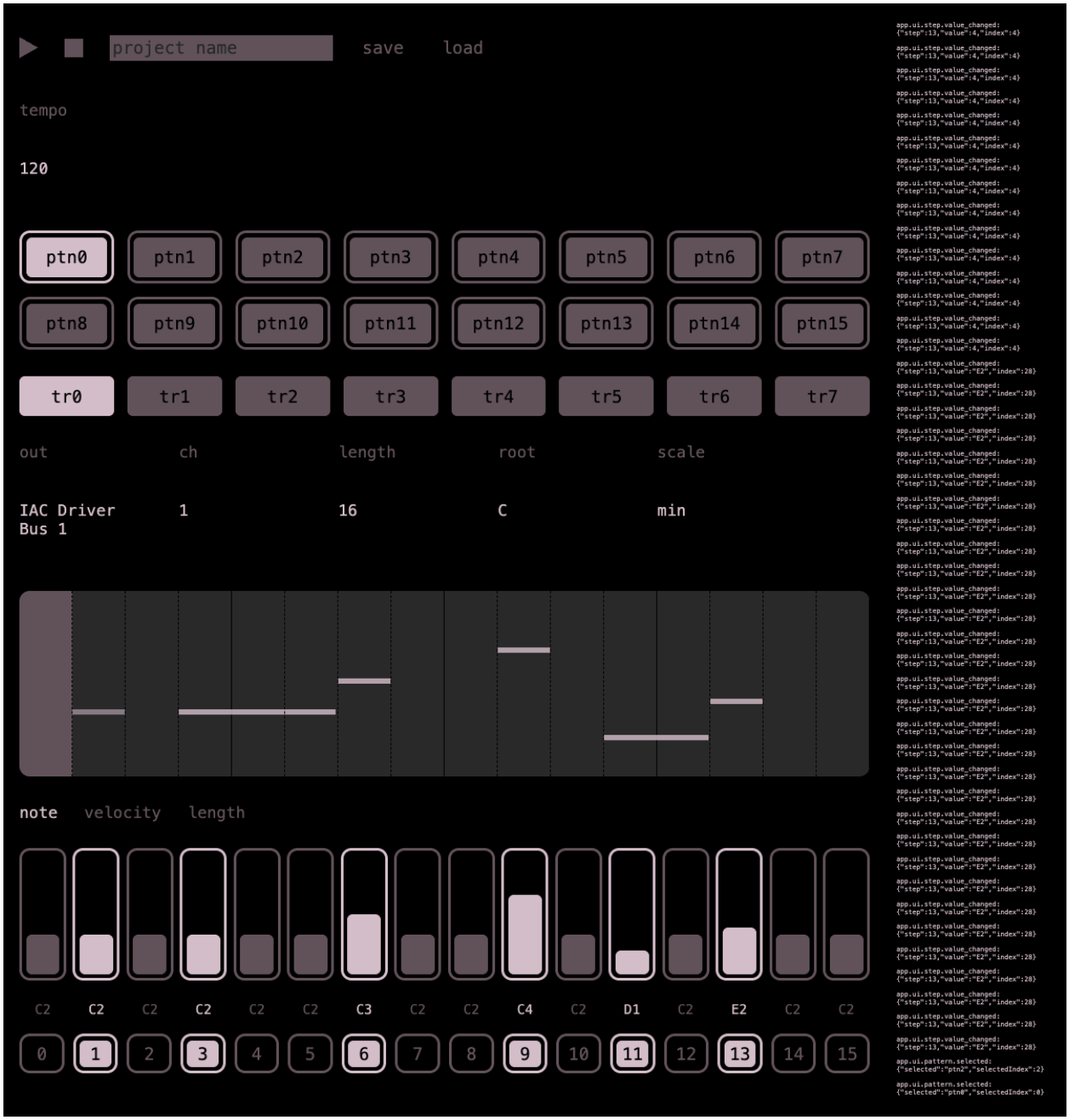

Рисунок 4.4 – Повністтю реалізований інтерфейс застосунку

4.3 Реалізація компонентів логіки та взаємодії з MIDI

Після того як логіка інтерфейсу та його відображення реалізована, маємо можливість почату роботу над самою системою. В архітектуру застосунку входять такі компоненти: інтерфейс, дані, конфігурації, метроном, секвенсор, та MIDI. Компонент інтерфейсу вже реалізований, а

дані які зберігає додаток будуть сформовані природою операцій доступних користувачу. Тож у цьому розділі будуть описані реалізації компонентів системи подій, метронома, управління секвенсором та інтеграції з Web MIDI API.

4.3.1 Система подій

Система подій в архітектурі формує головний тунель комунікацій між компонентами архітектури. Наприклад, коли відлік метронома крокує вперед, система повинна знати, що послідовність рухається далі і відповідні ноти повинні програвати(рисунок 4.5). Дії користувача теж повинні викликати зміни стану застосунку, робити відповідні модифікації даних, які обробляє MIDI та система в цілому. З оглядом на комплексність інтерфейсу та взаємодії між компонентами система подій повинна працювати окремо, незалежно від компонентів, а компоненти кожен окремо можуть прив’язуватися до подій, не втручаючись в роботу інших частин системи.

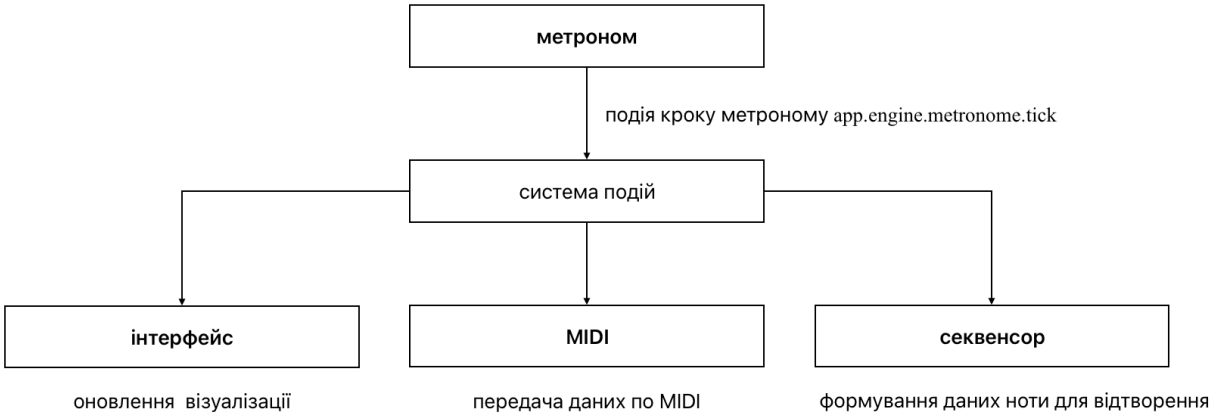

Рисунок 4.5 – Схема реакції системи на подію модуля Метроному

Ми вже реалізували деякі події інтерфейсу в класах елементів керування, але є події які не пов’язані з діями користувача. Для ефективного прослуховування подій вони повинні мати характерні назви, щоб відрізняти

їх одне від одного. Повний список подій системи буде виглядати наступним чином:

Таблиця 4.1 – Список подій зарезервованих в системі

Назва події	Опис
app.ui.start_button.click	запуск відтворення послідовності
app.ui.stop_button.click	зупинка відтворення
app.ui.tempo_scroll.scroll	зміна темпу користувачем
app.ui.pattern.selected	вибір послідовності (pattern)
app.ui.track.selected	вибір доріжки (track)
app.ui.track.midi_out_scroll.scroll	зміна MIDI пристрою для доріжки
app.ui.track.midi_ch_scroll.scroll	зміна MIDI каналу для доріжки
app.ui.track.length_scroll.scroll	зміна довжини доріжки
app.ui.step_value_control.selected	вибір типу редагованої властивості (нота, velocity, довжина)
app.ui.step.value_changed	зміна значення редагованої властивості ноти
app.ui.step.activated / app.ui.step.disabled	активація або деактивація ноти
app.engine.metronome.tick	кожен крок метроному, використовується для синхронізації
app.sequencer.note_played	подія, яка сигналізує, що нота має бути зіграна (і передана на MIDI-вихід)
app.ui.start	ініціалізація інтерфейсу
app.engine.initialized	ініціалізація MIDI-пристроїв

Для реєстрації подій та відповідних реакцій них, було реалізовано функцію App.bindEvents, яка прив’язує виклик функцій на елементах, відповідно до назви події.

Лістинг 4.24 – Функція bindEvents, яка реєструє усі події системи

```
  bindEvents: function() {
    // metronome
    this.Utils.listen('app.engine.metronome.tick', e => {
      this.UI.sequenceOverview.advanceCursor()
      this.Engine.sequencer.advance()
    });

    // start/stop
    this.Utils.listen('app.ui.start_button.click', e => {
      this.Engine.metronome.start()
    });

    this.Utils.listen('app.ui.stop_button.click', e => {
      this.Engine.metronome.stop()
      this.UI.sequenceOverview.resetCursor()
    });

    // tempo
    this.Utils.listen('app.ui.tempo_scroll.scroll', e => {
      const newTempo = parseInt(e.detail.value)

      this.currentPattern().tempo = newTempo;
      this.Engine.metronome.setTempo(newTempo);
      this.UI.update();
    });

    // pattern selection
    this.Utils.listen('app.ui.pattern.selected', e => {
      this.setCurrentPattern(e.detail.selectedIndex)
      this.UI.update();
    });

    // track selection
    this.Utils.listen('app.ui.track.selected', e => {
      this.setCurrentTrack(e.detail.selectedIndex)
      this.UI.update();
    });

    // track controls
    this.Utils.listen('app.ui.track.midi_out_scroll.scroll', e
=> {
      this.currentTrack().midiOut = e.detail.value;
      this.UI.update();
    })
    this.Utils.listen('app.ui.track.midi_ch_scroll.scroll', e
=> {
      this.currentTrack().midiChannel = e.detail.value;
      this.UI.update();
    })
    this.Utils.listen('app.ui.track.length_scroll.scroll', e
=> {
```

```
        this.currentTrack().length = e.detail.value;
        this.UI.update();
      })
      this.Utils.listen('app.ui.track.root_note_scroll.scroll',
e => {
        this.currentTrack().rootNote = e.detail.value;
        this.UI.update();
      })
      this.Utils.listen('app.ui.track.scale_scroll.scroll', e =>
{
        this.currentTrack().scale = e.detail.value;
        this.UI.update();
      })

      // step controls change
      this.Utils.listen('app.ui.step_value_control.selected', e
=> {
        const selected = e.detail.selected

        if (this.data.activeValueControls == selected) return;

        this.data.activeValueControls = selected
        this.UI.update()
      });

      // step value change
      this.Utils.listen('app.ui.step.value_changed',
this.Utils.debounce(e => {
        const step = this.currentTrack().steps[e.detail.step];
        const value = e.detail.value;
        const index = e.detail.index

        step[this.data.activeValueControls] = value
        this.UI.update()
      }, 10));

      // step toggle
      this.Utils.listen('app.ui.step.activated', e => {
        const step = e.detail.step
        this.UI.stepValueControls[step].toggleActive(true)
        this.currentTrack().steps[step].active = true
        this.UI.update()
      });

      this.Utils.listen('app.ui.step.disabled', e => {
        const step = e.detail.step
        this.UI.stepValueControls[step].toggleActive(false)
        this.currentTrack().steps[step].active = false
        this.UI.update()
      });

      this.Utils.listen('app.engine.initialized', e => {
        Array.from(e.detail.outputs).forEach(out => {
```

```
        this.Utils.log('MIDI device found:', out.name)
      })
    })

    this.Utils.listen('app.sequencer.note_played', e => {
      const note = e.detail
      this.Engine.adapter.send(note)
    })
  },
```

Цей код має багато функцій, які ми ще не реалізували, але для повної картини важливо це продемонструвати. Окремо, відзначимо функцію App.UI.update(), яка викликається що разу коли інтерфейс повинно бути оновлено. Головною задачею функції App.UI.update() є перезавантаження коду інтерфейсу, який ми реалізували у попередньому розділі. Багато обробників подій викликають вже зареєстровані елементи інтерфейсу(наприклад, this.UI.stepValueControls[step].toggleActive(false) отримує певний елемент активації ноти, та змінює її стан на вимкнений).

4.3.2 Рівень даних

Рівень даних реалізуємо у вигляді вкладеної структури, яка формується у внутрішньому об’єкті App.data. Цей об’єкт містить усю інформацію про поточний стан музичного проекту та параметри, з якими працює користувач у реальному часі. Усі зміни, які відбуваються через інтерфейс користувача, безпосередньо змінюють відповідні властивості в цій структурі за допомогою системи подій. Це забезпечує реактивність: інтерфейс оновлюється відповідно до стану App.data функцією App.UI.update().

У центрі структури є поняття збережених послідовностей або патернів (patterns), кожен з яких містить набір треків (tracks). Кожен трек, у свою чергу, складається з фіксованої кількості кроків (steps), що відповідають окремим нотам. Для кожного кроку зберігається інформація про:

- висоту ноти (наприклад, C4)
- інтенсивність (velocity)

- тривалість звучання (length)
- активність (чи буде ця нота зіграна)
- обраний MIDI канал (1–16)
- вихідний MIDI пристрій
- довжину послідовності (4–16 кроків)

Також об’єкт зберігає загальний стан застосунку: номер активного патерну, обраний трек, активний режим редагування (наприклад, зміна висоти нот або гучності), темп відтворення.

Лістинг 4.25 – Приклад структури даних в процесі використання

```
{
  "tempo": 120,
  "currentPattern": 0,
  "currentTrack": 0,
  "patterns": [
    {
      "tracks": [
        {
          "midiOut": 0,
          "midiChannel": 1,
          "length": 16,
          "steps": [
            {
              "active": true,
              "step": 0,
              "note": "C2",
              "velocity": 100,
              "length": 1
            },
            {
              "active": false,
              "step": 1,
              "note": "D2",
              "velocity": 80,
              "length": 1
            },
            {
              "active": true,
              "step": 2,
              "note": "E2",
              "velocity": 110,
              "length": 2
            }
            // ... 16 кроків
          ]
```

```
        }
        // ... треків
      ]
    }
    // ... патернів
  ]
}
```

Додатково, конфігураційний модуль у застосунку відповідає за зберігання усіх початкових параметрів, які визначають базову поведінку системи та початкові значення для користувацьких налаштувань.

- кількість доступних патернів і треків
- допустимі значення нот, velocity, довжини кроків
- доступний темп
- значення за замовчуванням для темпу, довжини патерна, основної ноти та масштабу;
- список підтримуваних MIDI-каналів і пристроїв;

Фактично, конфігураційний модуль є центральною точкою налаштувань, яка дозволяє швидко адаптувати застосунок до нових потреб без зміни логіки коду. Він також спрощує ініціалізацію даних при запуску проекту.

Для зручної роботи з даними, реалізуємо наступні функції:

- generatePattern() - генерує повноцінний патерн із 8 доріжками. Кожна з них створюється функцією generateTrackPattern().
- resetPatternsData() - повністю ініціалізує всю ієрархію даних – масив з 16 патернів, кожен з яких містить 8 доріжок, кожна з 16 нотами.
- currentPattern() - повертає поточний активний патерн відповідно до індексу currentPattern у глобальному стані.
- currentTrack() - повертає обрану доріжку (track) з поточного патерна.
- setCurrentPattern(index) / setCurrentTrack(index) - змінюють активний патерн або трек, оновлюючи відповідний індекс у стані.

Логіка зміни та отримання даних з використанням цих функцій була описана обробкою подій в попередньому розділі

4.3.3 Метроном

Головним рушієм послідовності є компонент Метроному. Його задача робити точний відлік часу, базуючись на конфігурації даних темпу та посилати подію «app.engine.metronome.tick». Подія буде потім перехоплена інтерфейсом для візуалізації шагу послідовності, та, найголовніше, вона буде перехоплена нашим секвенсором, який формує що програвати у даний момент.

Головною технічною особливістю компоненту метроному є точність, яка повинна бути гарантовано алгоритмом для ідеальної синхронізації між музичними пристроями. Базова JavaScript функція setInterval[9], на жаль, не гарантує точності часу програвання, оскільки вона оптимізована для роботи з документом у браузері. Натомість, реалізація базується на інтерфейсі HTML AudioContext.currentTime, що забезпечує високу точність синхронізації порівняно зі звичайними таймерами JavaScript. Метроном створює аудіоконтекст, який синхронізує момент удару з часом, що дозволяє уникати дрібних затримок, притаманних системному setInterval. Замість того, щоб запускати подію в певний момент часу, компонент заздалегідь обчислює наступний крок метроному і розписує майбутні удари[10]. Цикл перевіряє, чи настав час для чергового удару, і якщо так - викликає подію «app.engine.metronome.tick», що потім використовується іншими компонентами, як-от секвенсор. Важливо відмітити, що метроном посилає сигнали не з частотою параметра tempo(ударів на хвилину), а натомість ділить кожен удар на 4 subDivision. Тобто, якщо темп встановлений на позначку 120, то метроном зробить 120 * 4 = 480 ударів на хвилину, бо це важливо для деталізації музичних послідовностей.

Клас Metronome є частиною App.Engine, що містить всю внутрішню логіку застосунку. Вхідний параметр класу - темп, тобто кількість ударів на хвилину, а функціями керування будуть: setTempo() для зміни темпу в реальному часі, start() та stop().

Лістинг 4.26 – Код модуля Метроному, як частина Engine

```
Metronome: class Metronome extends EventTarget {
   constructor(tempo = 120) {
     super();

     this.audioCtx = new (window.AudioContext ||
window.webkitAudioContext)();
     this.tempo = tempo;

     this.mainInterval = 60 / this.tempo;
     this.subdivision = 4;
     this.interval = this.mainInterval / this.subdivision;

     this.nextTickTime = 0;
     this.lookahead = 25; // ms
     this.scheduleAheadTime = 0.1; // seconds
     this.timerID = null;
     this.isRunning = false;

     this.tickCount = 0;
   }

   start() {
     if (this.isRunning) return;
     this.isRunning = true;

     this.audioCtx.resume();
     this.nextTickTime = this.audioCtx.currentTime + 0.1;
     this.scheduler();
   }

   stop() {
     this.isRunning = false;
     if (this.timerID) clearTimeout(this.timerID);
     this.tickCount = 0;
   }

   scheduler() {
     while (this.nextTickTime < this.audioCtx.currentTime +
this.scheduleAheadTime) {
       this.scheduleTick(this.nextTickTime);
       this.nextTickTime += this.interval;
     }

     this.timerID = setTimeout(() => this.scheduler(),
this.lookahead);
   }

   scheduleTick(time) {
     const isMainClick = this.tickCount % this.subdivision
=== 0;
```

```
          document.dispatchEvent(new
CustomEvent('app.engine.metronome.tick', {
            detail: {
              time,
              tempo: this.tempo,
              beat: this.tickCount / this.subdivision,
              tick: this.tickCount
            }
          }));

          this.tickCount++;
        }

        setTempo(newTempo) {
          this.tempo = newTempo;
          this.mainInterval = 60 / this.tempo;
          this.interval = this.mainInterval / this.subdivision;
        }

        calculateDuration(bars) {
          return((60000 / this.tempo) / 4) * bars
        }
      },
```

4.3.4 Вибір пристроїв, маршрутизація

У застосунку вибір MIDI пристроїв та їхня маршрутизація реалізується через модуль MIDI, що є частиною App.Engine. Цей модуль відповідає за:

- Підключення до MIDI-пристроїв. Під час ініціалізації застосунок викликає метод navigator.requestMIDIAccess(), щоб отримати доступ до системних MIDI-входів та виходів. Успішне підключення зберігає список пристроїв у масивах inputs та outputs.

- Оновлення списку пристроїв. У разі зміни стану підключення (наприклад, користувач під’єднав або вимкнув пристрій) - генерується подія «app.engine.initialized», що автоматично оновлює список доступних пристроїв.

Лістинг 4.27 – Код реєстрації події знайдених підключених пристроїв

```
this.Utils.listen('app.engine.initialized', e => {
  Array.from(e.detail.outputs).forEach(out => {
    this.Utils.log('MIDI device found:', out.name)
```

```
    })
  })
```

- Для кожної доріжки (track) зберігається індекс обраного вихідного пристрою. Він вказує, на який саме фізичний або віртуальний MIDI-вихід буде надсилатися повідомлення. Це дозволяє призначати різні пристрої на різні треки.

- Вибір MIDI-каналу. Кожна доріжка також має параметр midiChannel – від 1 до 16, відповідно до стандарту MIDI. Це дає можливість одночасної роботи з кількома інструментами навіть через один вихід.

- Надсилання повідомлень. Під час відтворення, оброблені події надсилаються через метод send() у MIDI модулі, який будує MIDI-повідомлення noteOn та noteOff згідно з параметрами кроку послідовності(нота, velocity, канал, довжина тощо) і надсилає їх на відповідний output через Web MIDI API[2]. Вартно відмітити метод createMessage, який формую MIDI повідомлення формуючи байти статусу та даних, згідно до формату протоколу MIDI[1], який ми розглядали раніше.

Лістинг 4.28 – Код модуля MIDI, як частина Engine

```
MIDI: class {
   static noteMap = {
     'C': 0, 'C#': 1,
     'D': 2, 'D#': 3,
     'E': 4,
     'F': 5, 'F#': 6,
     'G': 7, 'G#': 8,
     'A': 9, 'A#': 10,
     'B': 11,
   };

   constructor() {
     this.access = null;
     this.inputs = [];
     this.outputs = [];
   }

   async connect() {
     try {
       this.access = await navigator.requestMIDIAccess();
       this.updateDeviceList();
```

```
            this.access.onstatechange = async () => {
              this.access = await navigator.requestMIDIAccess();
              this.updateDeviceList();

              document.dispatchEvent(new
CustomEvent('app.engine.initialized', {
                details: { outputs: this.outputs }
              }));
            };
          } catch (err) {
          }
        }

        updateDeviceList() {
          if (!this.access) return;

          this.inputs = Array.from(this.access.inputs.values());
          this.outputs =
Array.from(this.access.outputs.values());
        }

        outputNames() {
          return this.outputs.map((input) => {
            return input.name
          })
        }

        send(note) {
          const message = this.createMessage(note)
          const output = this.outputs[note.midiOut]

          if (!output) {
            console.log('no output found')
            return;
          }

          const { noteOn, noteOff } = this.createMessage(note);
          const time = note.time || performance.now();

          output.send(noteOn, time);
          output.send(noteOff, time + note.duration);
        }

        createMessage(note) {
          const noteValue = this.noteToValue(note.note);
          const channel = note.midiChannel || 1;
          const statusOn = 0x90 + (channel - 1);
          const statusOff = 0x80 + (channel - 1);

          const noteOn = [statusOn, noteValue, note.velocity];
          const noteOff = [statusOff, noteValue, 0];
```

```
          return { noteOn, noteOff };
        }

        noteToValue(noteName) {
          const match = noteName.match(/^([A-Ga-g]#?|[A-Ga-
g]b?)(-?\d+)$/);
          if (!match) return null;

          const [, pitch, octaveStr] = match;
          const semitone =
App.Engine.MIDI.noteMap[pitch.toUpperCase()];
          const octave = parseInt(octaveStr, 10);

          return (octave + 1) * 12 + semitone;
        }
      }
```

4.3.5 Секвенсор та MIDI

Компонент Секвенсору реалізований як клас App.Engine.Sequencer. Sequencer - це механізм, який на кожному кроці метроному перевіряє активні ноти в доріжках(tracks) активного патерну(pattern) та генерує відповідні події для відтворення. Секвенсор поєднується з MIDI через систему подій. Замість безпосередньої роботи з MIDI, Секвенсор лише створює подію «app.sequencer.note_played» з усіма параметрами: note, velocity, length, midiOut, channel, time. Інші модулі (наприклад, MIDI) підписуються на цю подію та вже відправляють повідомлення через Web MIDI API.

Лістинг 4.29 – Код реєстрації події програшу ноти

```
this.Utils.listen('app.sequencer.note_played', e => {
  const note = e.detail
  this.Engine.adapter.send(note)
})
```

Метод App.Engine.Sequencer.advance() викликається при кожному удару метроному (подія «app.engine.metronome.tick»), що забезпечує точне розміщення нот у часі. В модулі є лічильник, який відповідає довжині доріжки, та скидається в 0, коли досягне кінця доріжки, щоб грати по колу, поки користувач не натисне стоп.

Лістинг 4.30 – Код модуля Sequencer, як частина Engine

```
 Sequencer: class {
   constructor(tempo, tracks = [], tickDivisions = 4) {
     // this.tempo = tempo;
     this.stepCounter = 0;
     this.barDuration = (60000 / tempo) / tickDivisions
     this.tracks = tracks;
   }

   advance() {
     const now = performance.now();

     this.tracks.forEach((track, trackIndex) => {
       const currentStepIndex = this.stepCounter %
track.length;
       const step = track.steps[currentStepIndex];

       if (step.active) {
         document.dispatchEvent(new
CustomEvent('app.sequencer.note_played', {
           detail: {
             note: step.note,
             velocity: step.velocity,
             duration: step.length * this.barDuration,
             midiOut: track.midiOut,
             midiChannel: track.midiChannel,
             time: now,
             trackIndex,
             currentStepIndex
           }
         }));
       }
     })

     this.stepCounter++;
   }

   reset() {
     this.stepCounter = 0;
   }
 },
```

4.3.6 Додаткові функції

В додатку буде реалізовано модуль Utils, який буде містити додаткові функції для синхронізації між компонентами та інші другорядні операції:

Функція App.Utils.debounce() запобігає надто частому виклику

обробника події, якщо він запускається багато разів підряд (наприклад, під час перетягування чи скролу). Функція викликає обробник лише після вказаної затримки з попереднього виклику. Це допомагає видалити зайві проміжні виклики обробників подій. Наприклад, коли змінюємо темп з 80 до 120, обробник буде викликано менше ніж 40 разів.

Лістинг 4.31 – Код функції видалення зайвих викликів подій

```
debounce: function (f, delay) {
  let timer;
  return function (...args) {
    clearTimeout(timer);
    timer = setTimeout(() => f.apply(this, args), delay);
  };
},
```

Функція App.Utils.listen() полегшує підписку на події. При отриманні події виконує передану функцію, попередньо прив’язавши її до контексту App. Також автоматично записує події до моніторингу подій.

Лістинг 4.32 – Код функції видалення зайвих викликів подій

```
listen: function(eventId, f) {
  document.addEventListener(eventId, function(e) {
    f.bind(App)(e);
    App.Utils.log(eventId, e.detail);
  })
},
```

Функція App.Utils.log() виводить у консоль та в UI повідомлення про події, що відбулися в застосунку.

Лістинг 4.33 – Код функції видалення зайвих викликів подій

```
log: function (e, detail) {
  console.log(`event fired: ${e}`);
  App.UI.monitoringLog.log(e, detail);
}.bind(this)
```

Наш UI має список подій, які в реальному часі додаються до елементу моніторингу подій. За цей елемент відповідає клас MonitoringLog.

Лістинг 4.34 – Код модуля списку подій, як частина Engine

```
MonitoringLog: class {
  constructor(el) {
    this.el = el;
    this.maxItems = 48;
  }

  log(e, obj) {
    const logItem = document.createElement('div');
    logItem.className = 'log-item';
    logItem.textContent = `${e}: ${JSON.stringify(obj)}`;

    this.el.appendChild(logItem);

    while (this.el.children.length > this.maxItems) {
      this.el.removeChild(this.el.firstChild);
    }
  }
},
```

У підсумку, архітектура застосунку побудована як набір взаємодіючих модулів, кожен з яких виконує чітко визначену функцію. Конфігураційний модуль(App.config) формує основу для всієї логіки, задаючи параметри, з якими працює система. Модуль даних(App.data) відповідає за збереження поточного стану, структуру патернів, доріжок і користувацьких змін. Метроном(App.Engine.Metronome) забезпечує точне ритмічне ядро для синхронного відтворення подій. MIDI модуль реалізує зв’язок із зовнішніми пристроями, формує й надсилає повідомлення відповідно до обраних параметрів. Логіка музичних послідовностей(App.Engine.Sequencer) керує програванням і логікою активних нот у часі. Графічний інтерфейс(App.UI) синхронізує візуальне відображення із внутрішнім станом і забезпечує зручну взаємодію користувача із системою. Разом ці модулі формують гнучку, розширювану архітектуру веб MIDI контролеру.

5 ТЕСТУВАННЯ ТА ВІДЛАГОДЖЕННЯ СИСТЕМИ

На стадії проектування застосунку було розроблено сценарій користувача, який ми повинні протестувати та переконатися що застосунок працює відповідно до вимог. Виконуючи сценарій крок за кроком ми проаналізуємо які частини системи працюють коректно, а які потребують відлагодження та покращення.

Отже, попередні умови сценарію:

1. До комп’ютера підключено хоча б один MIDI-пристрій.

Для тестування обрано апаратний музичний синтезатор Elektron Digitakt, якій реалізує MIDI протокол у USB. Тож підключаємо пристрій до комп’ютера за допомогою USB кабелю, та переконаємося що він відображається в списку підключених пристроїв ОС.

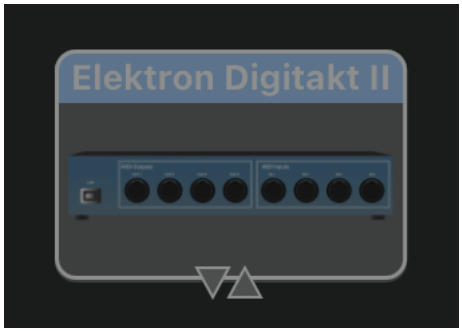

Рисунок 5.1 – Пристрій знайдено в ОС

2. Веб-застосунок запущено у сумісному браузері з підтримкою Web MIDI API.

Для розробки ми обрали Google Chrome, який є повністю сумісним зі стандартом Web MIDI API.

3. Доступ до MIDI-портів дозволено користувачем.

При завантаженні застосунку у браузер, з’являється модальне вікно дозволу на доступ до підключених MIDI пристроїв. Погоджуємося та гарантуємо доступ браузеру до пристроїв.

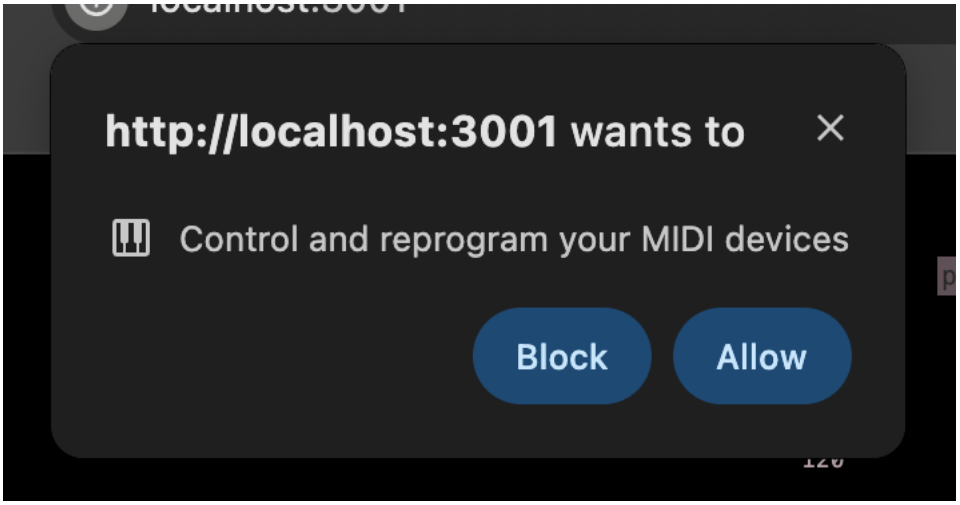

Рисунок 5.2 – Модально вікно доступу до MIDI пристроїв

Коли попередні умови виконані, ми можемо почати тестування застосунку, крок за кроком виконуючі пункти сценарію:

1. Користувач відкриває веб-застосунок.

Застосунок вже відкритий в браузері і повністю відображається коректно.

2. Надає дозвіл Web MIDI API на доступ до пристроїв.

Доступ до пристроїв надано як попередня умова сценарію.

3. Обирає один з доступних MIDI пристроїв.

Переконаємося що наш підключений пристрій(Elektron Digitakt) відображається у списку MIDI виходів(midi out).

Рисунок 5.3 – Підключений пристрій у списку пристроїв додатку

Застосунок дійсно знаходить пристрій, він відображається у списку. Тож переконаємося що саме він обраний як пристрій MIDI виходу.

4. Обирає MIDI-канал для треку.

Обираємо номер MIDI каналу(midi channel) який відрізняється від номеру по замовчуванню(номер MIDI каналу по замовчуванню - 1), наприклад, 4.

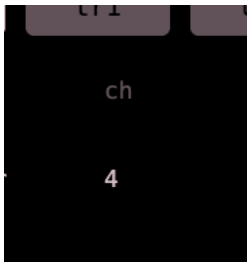

Рисунок 5.4 – Обраний MIDI канал в інтерфейсі користувача

5. Налаштовує темп.

Обираємо темп композиції(tempo) який відрізняється від темпу по замовчуванню(по замовчуванню - 120), наприклад, 158.

Рисунок 5.5 – Обраний темп музичної послідовності в інтерфейсі користувача

6. Увімкнувши активні кроки, задає їм ноти, інтенсивність та довжину.

Вводимо просту музичну послідовність з 5 нот. Виставляємо їм інтенсивність

звучання(velocity) та тривалість(length).

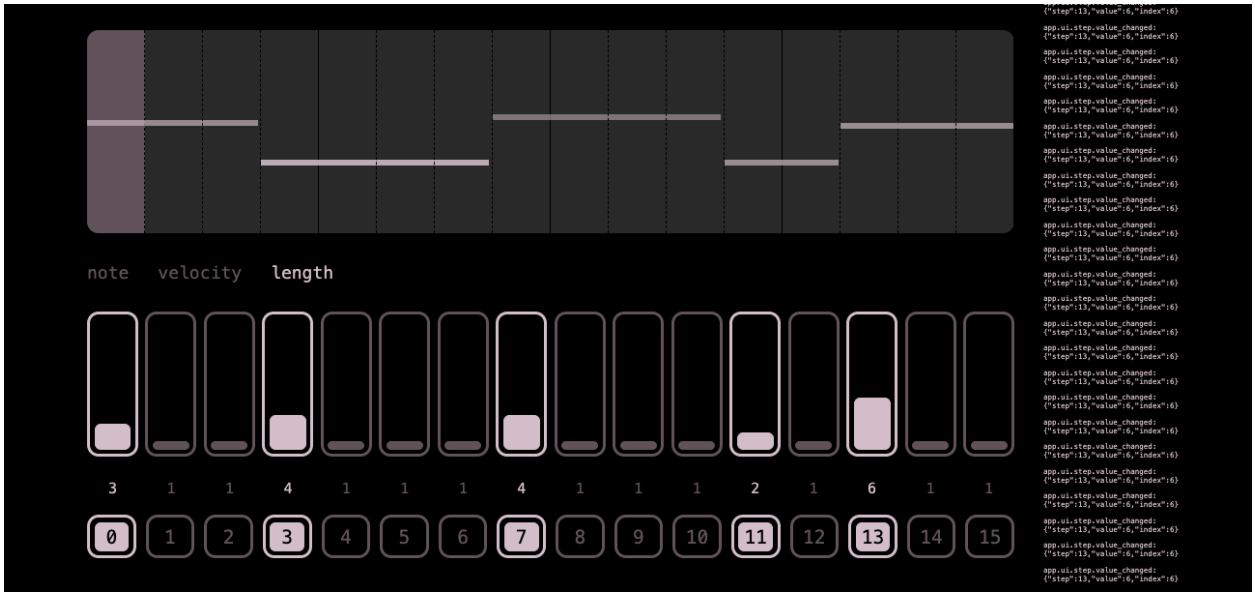

Рисунок 5.6 – Проста музична послідовність в інтерфейсі користувача

7. Натискає Start – метроном запускає відтворення.

При натисканні кнопки «Start», переконаємося що відтворення послідовності дійсно починається з першої ноти до останньої, а потім починається програватися з початку по колу.

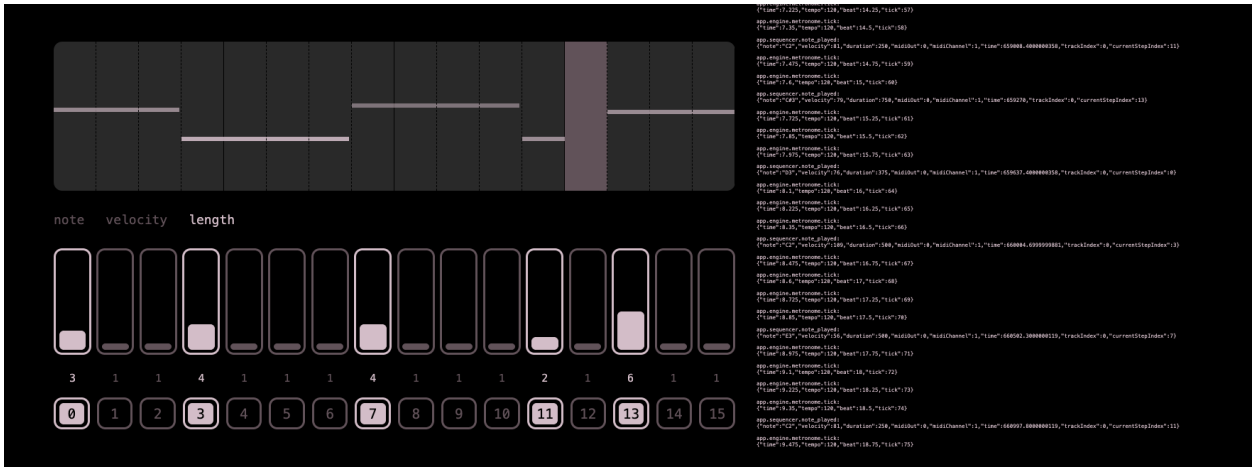

Рисунок 5.6 – Прогресія віддтворення музичної послідовності

Як бачимо, курсор послідовності рухається зі швидкістю обраного темпу, рухаючись по нотам послідовності зліва направо. Додатково, ми бачимо що у списку моніторингу справа з’являються події «app.engine.metronome.tick», що

свідчить про роботу метроному, та «app.sequencer.note_played», що свідчить про відтворення нот, доданих у послідовність.

8. Секвенсор грає активні ноти, передаючи їх у MIDI-пристрій.

Переконаємося що пристрій дійсно отримує передані ноти. Спочатку пристрій не реагував на дії застосунку. Після додаткових налаштувань на пристрої було обрано правильний MIDI канал, що співпадає з обраним каналом(midi channel) у застосунку - канал 4. Тепер бачимо що наш пристрій звучить та відтворює коректну послідовність нот.

9. Користувач не зупиняючи відтворення змінює параметри в реальному часі.

Важливою характеристикою нашого застосунку є можливість зміни послідовності у реальному часі та без зупинки відтворення. Тож змінемо послідовність на довільну та переконаємося що пристрій зчитує зміни і відтворює правильні ноти

Як додатковий сценарій, повторимо сценарій на інших MIDI каналах пристрою, обираючи наступні доріжки(track) та створюючи нові послідовності та переконаємося що система як треба відпрацьовує музичні патерни.

ВИСНОВКИ

У результаті роботи було розроблено веб застосунок, що виконує роль інтерфейсу керування MIDI-пристроями, що підключені до комп’ютера користувача. Такий підхід забезпечує мобільність у роботі з апаратним музичним обладнанням без потреби встановлювати спеціалізоване ПЗ.

Завдяки підтримці Web MIDI API, система дозволяє створювати, редагувати та запускати MIDI-послідовності, налаштовувати канали, параметри нот і маршрутизацію сигналів до вибраних пристроїв у режимі реального часу. Усе це реалізовано через інтуїтивно зрозумілий графічний інтерфейс, доступний з будь-якого сумісного браузера.

Серед головних переваг рішення – кросплатформеність, відсутність необхідності у встановленні програм, швидкий доступ до функцій секвенсора, можливість збереження та завантаження проектів. Такий застосунок може бути особливо корисним для музикантів, викладачів або учнів, які працюють з MIDI-обладнанням у різних умовах.

Водночас система має низку обмежень: залежність від браузера й комп’ютера, часткова підтримка Web MIDI API в деяких середовищах, обмежений функціонал порівняно з професійними цифровими аудіо станціями. Проте, як концепція веб контролера – це ефективне рішення, що може бути основою для подальшого розвитку, масштабування або інтеграції з іншими музичними сервісами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. MIDI 1.0. Core Specification. The MIDI Association [Електронний ресурс]. – Режим доступу: https://midi.org/midi-1-0-core-specifications

2. Web MIDI API - Web APIs | MDN. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_MIDI_API

3. Creative tools for music makers | Ableton [Електронний ресурс]. – Режим доступу: https://www.ableton.com/

4. SQ-1 Owner’s Manual | KORG(USA). [Електронний ресурс]. – Режим доступу: https://www.korg.com/us/support/download/manual/0/436/2017/

5. Web MIDI Api | Caniuse Support tables for HTML5, CSS3, etc. [Електронний ресурс]. – Режим доступу: https://caniuse.com/midi

6. Elektron Digitakt 2: User Manual, OS 1.10B. [Електронний ресурс]. – Режим доступу: https://www.elektron.se/wp-content/uploads/2025/06/Digitakt-2-User-Manual_ENG_OS1.10B_250609.pdf

7. Event-driven architecture – Wikipedia. [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Event-driven_architecture

8. Basic concepts of flexbox - CSS | MDN. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox

9. Window: setInterval() method - Web APIs | MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en/docs/Web/API/Window/setInterval

10. GitHub - cwilso/metronome: Web Audio metronome example to show scheduling. [Електронний ресурс]. – Режим доступу: https://github.com/cwilso/metronome