

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система обробки партитур вокальних синтезаторів»

Виконав студент групи ПЗп-21-4
САВЧЕНКО Ерік Олександрович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«06» червня 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **САВЧЕНКО Ерік Олександрович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Система обробки партитур вокальних синтезаторів»**,
затверджена Наказом закладу вищої освіти від **«06» червня 2026 р., №329**

Вихідні дані для роботи: **партитури вокальних синтезаторів, навчальна література, вебджерела, статті, документації**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	06.06.2026р.	06.06.2026р.
2	Єфремов Ю.М.	06.06.2026р.	06.06.2026р.
3	Єфремов Ю.М.	06.06.2026р.	06.06.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку для обробки партитур вокальних синтезаторів та пояснювальної записки. Пояснювальна записка до випускної роботи містить 69 сторінок, 27 ілюстрацій, 5 таблиць, 30 джерел та додатки.

Метою роботи є розширення сумісності цифрових партитур вокальних синтезаторів шляхом обробки та конвертації методом приведення даних до уніфікованого формату.

Розроблена система для обробки партитур вокальних синтезаторів забезпечує високу точність та ефективність конвертації між різними форматами партитур, а також можливість автоматичної заміни лірики та транспонування нот. Система являє собою вебдодаток, що забезпечує легкий доступ для користувачів, а система плагінів дозволяє легко додавати підтримку нових форматів партитур.

КЛЮЧОВІ СЛОВА: КОНВЕРТАЦІЯ, ВОКАЛЬНІ СИНТЕЗАТОРИ, ФАЙЛИ ПАРТИТУР

					ІПЗ.КР.Б. – 121 – 26 – ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Е.О. Савченко			Система обробки партитур вокальних синтезаторів	Літ.	Арк.
Керівник		Ю.М. Єфремов					Аркушів
							3 63
Рецензент		О.В. Власенко				Житомирська політехніка, група ІПЗп-21-4	
Зав. каф.		Т.А. Вакалюк			Пояснювальна записка		

ABSTRACT

The bachelor's qualification thesis consists of a web application for processing vocal synthesizer scores and an explanatory note. The explanatory note to the thesis contains 69 pages, 27 illustrations, 5 tables, 30 sources and appendix.

The purpose of the work is to ensure the compatibility of score files of vocal synthesizers by processing and converting them by the method of bringing data to a unified format.

The developed system for processing scores of vocal synthesizers provides high accuracy and efficiency of conversion between different formats of scores, as well as the ability to automatically replace lyrics and transpose notes. The system is a web-application that provides easy access for users, and the plugin system allows you to easily add support for new formats of scores.

KEYWORDS: CONVERSION, VOCAL SYNTHESIZERS, SCORE FILES

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА	
ЗАДАЧІ	10
1.1 Загальна характеристика предметної області.....	10
1.2 Аналіз аналогів програмного продукту.....	12
1.3 Аналіз форматів файлів партитур.....	16
1.4 Обґрунтування вибору архітектурного підходу та інструментальних засобів.....	22
Висновки до першого розділу.....	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ СТРУКТУРИ ДАНИХ ТА АРХІТЕКТУРИ	
ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	27
2.1 Дизайн архітектури та проєктування системи.....	27
2.2 Функціональні та нефункціональні вимоги до системи.....	30
2.3 Канонічна модель та конвеєр конвертації.....	33
2.4 Архітектура плагінів та реєстр конвертерів.....	37
2.5 Реалізація веб-інтерфейсу.....	41
2.6 Реалізація проміжних перетворень.....	44
2.7 Аналіз реалізованих плагінів форматів.....	47
Висновки до другого розділу.....	50
РОЗДІЛ 3. ПРИКЛАД РОБОТИ ПРОГРАМИ	53
3.1 Інтерфейс користувача та порядок роботи з програмою.....	53
3.2 Тестування та демонстрація роботи програми.....	57
Висновки до третього розділу.....	64
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67
ДОДАТКИ	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CCS – формат файлів партитур програми CeVIO Creative Studio

JSON (JavaScript Object Notation) – текстовий формат обміну даними, зручний як для читання людиною, так і для машинної обробки

MIDI (Musical Instrument Digital Interface) – стандарт цифрового представлення музичних даних; визначає, зокрема, нумерацію нот за висотою

SVP – формат файлів партитур програми Synthesizer V Studio

UST – формат файлів партитур програми UTAU

USTx – формат файлів проєктів програми OpenUTAU

VPR – формат файлів проєктів програми VOCALOID (нових версій)

VSQX – формат файлів партитур програми VOCALOID версій 3–4

XML (eXtensible Markup Language) – розширювана мова розмітки для зберігання та передачі структурованих даних

YAML (YAML Ain't Markup Language) – людиночитаний формат серіалізації даних, що використовується у файлах USTx

ВСТУП

Актуальність теми. Протягом останніх двох десятиліть технології синтезу співочого голосу пройшли шлях від експериментальних розробок до повноцінного інструмента сучасної музичної індустрії. Вокальний синтезатор – це програмний засіб, що відтворює спів людського голосу штучно, на основі заданих користувачем нот та тексту: композитор «програмує» вокальну партію, а програма синтезує звучання, з'єднуючи заздалегідь записані фрагменти голосу реального співака або генеруючи звук за допомогою математичної моделі. Це принципово змінило процес створення вокальної музики, надавши авторам можливість працювати з голосом без участі живого виконавця.

Сьогодні вокальні синтезатори застосовуються в широкому колі сфер людської діяльності. У музичній індустрії на їх основі створено цілий культурний феномен віртуальних виконавців, найвідомішим з яких є Natsune Miku; продюсери використовують синтезатори для запису демонстраційних версій пісень, бек-вокалу та повноцінних вокальних партій. У сфері створення цифрового контенту синтезатори застосовують автори каверів, відеоблогери та розробники ігор і анімації для озвучення персонажів. В освіті такі засоби використовують для навчання вокалу, демонстрації принципів музичної теорії та інтерактивної взаємодії зі студентами. Окремо слід відзначити роль вокального синтезу у забезпеченні доступності – наданні синтезованого голосу людям, що його позбавлені.

Розвиток цієї галузі тісно пов'язаний із загальними тенденціями розвитку інформаційних технологій. Якщо перші системи ґрунтувалися на конкатенації (склеюванні) записаних зразків голосу, то сучасні рішення дедалі частіше використовують моделі машинного навчання та глибокі нейронні мережі, що суттєво підвищило природність звучання. Паралельно зростає роль веб-технологій: складні інструменти обробки даних, які раніше потребували встановлення, поступово переносяться у браузер, що робить їх доступнішими для широкого кола користувачів незалежно від платформи.

Питання синтезу співочого голосу досліджувалося як науковцями, так і розробниками програмного забезпечення. Основи комерційного вокального синтезу було закладено у спільному дослідницькому проєкті під керівництвом Хідекі Кенмоті в Університеті Помпеу Фабра, результатом якого стала технологія VOCALOID компанії Yamaha. Сучасний стан галузі узагальнено в оглядових наукових працях, присвячених системам синтезу співу на основі глибокого навчання [2, 3]. Водночас на ринку наявні готові програмні рішення для конвертації партитур між форматами – зокрема UtaFormatix, LibreSVIP та вбудовані засоби імпорту окремих синтезаторів (наприклад, Synthesizer V Studio).

Попри значний прогрес, у предметній області залишається коло невирішених важливих питань. Кожен вокальний синтезатор зберігає партитури у власному форматі файлу, через що проєкт, створений в одній програмі, не може бути безпосередньо відкритий в іншій. Це породжує проблему сумісності: користувачі, які працюють одночасно з кількома синтезаторами, змушені переносити проєкти вручну, витрачаючи значний час і ризикуючи втратою даних. Наявні засоби конвертації лише частково розв'язують цю проблему: одні складно розширювати на нові формати, інші спотворюють криву висоти тону під час конвертації між форматами з різним способом її зберігання, треті прив'язані до однієї екосистеми або потребують локального встановлення та некоректно працюють на окремих операційних системах. Таким чином, потреба у точному, розширюваному та платформонезалежному інструменті конвертації партитур залишається актуальною, що й зумовлює доцільність цієї роботи.

Мета роботи – розширення сумісності цифрових партитур вокальних синтезаторів шляхом їх обробки та конвертації методом приведення даних до уніфікованого проміжного представлення.

Завдання роботи:

1. Провести аналіз предметної області, визначити поняття вокального

синтезатора та партитури, виявити основні проблеми й потреби користувачів і сформулювати задачу з погляду інженерії програмного забезпечення (визначити входи та виходи системи).

2. Здійснити аналіз наявних програмних рішень-аналогів, виявити їх переваги та недоліки і сформулювати на цій основі вимоги до власного продукту.
3. Дослідити структуру файлів партитур поширених вокальних синтезаторів, визначити способи зберігання музичних даних у кожному форматі.
4. Спроекувати уніфіковану (канонічну) проміжну модель партитури, що агрегує музичні дані незалежно від конкретного формату.
5. Обґрунтувати вибір архітектурного підходу та інструментальних засобів і розробити програмне забезпечення для обробки та конвертації партитур.
6. Перевірити працездатність та коректність розробленої системи шляхом тестування й демонстрації її роботи.

Об'єкт дослідження – процес обробки та конвертації партитур вокальних синтезаторів, що породжує проблемну ситуацію несумісності форматів і охоплює зчитування, перетворення та запис музичних даних різних синтезаторів.

Предмет дослідження – методи та способи уніфікації й переносу музичної інформації між форматами партитур, зокрема способи приведення нот, темпу, тактового розміру, лірики та кривої висоти тону до єдиного проміжного представлення.

Методи дослідження – порівняльний аналіз структури файлів партитур різних форматів, доповнений методами проєктування моделей даних та об'єктно-орієнтованого проєктування для побудови уніфікованої моделі й розширюваної архітектури.

Отримані результати мають практичне значення для музикантів, продюсерів та розробників програмного забезпечення у сфері вокального синтезу, оскільки забезпечують розширення сумісності цифрових партитур та підвищення ефективності роботи з ними.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика предметної області

Вокальний синтезатор – це програмний засіб, що відтворює спів людського голосу штучно, на основі заданих користувачем нот та тексту. На відміну від звичайного музичного синтезатора, який генерує звуки інструментів, вокальний синтезатор формує саме вокальну партію: користувач вводить мелодію (ноти певної висоти й тривалості) та призначає кожній ноті склад тексту (лірику), а програма синтезує відповідний спів, з'єднуючи заздалегідь записані фрагменти голосу реального співака або генеруючи звук за допомогою математичної моделі.

Таким чином, робота з вокальним синтезатором нагадує роботу з нотним редактором: композитор не записує живий вокал, а «програмує» його. Це дозволяє створювати вокальні партії без участі співака, експериментувати з різними голосами та повністю контролювати кожен нюанс виконання – висоту, гучність, вібрато, вимову окремих фонем тощо.

Першим широковідомим комерційним рішенням став VOCALOID (компанія Yamaha, 2004 р.), технологія якого ґрунтувалася на конкатенації (склеюванні) зразків реального голосу [1]. Згодом з'явилися інші синтезатори – UTAU, CeVIO, Synthesizer V – частина з яких перейшла на моделі, побудовані на нейронних мережах [5-7]. Сьогодні вокальні синтезатори широко застосовуються як у професійній музичній індустрії, так і в аматорському середовищі, а також в освітніх та творчих проєктах.

Результат роботи користувача у вокальному синтезаторі зберігається у файлі партитури (англ. score file) – проєктному файлі, що містить усю інформацію, необхідну для відтворення вокальної партії. Партитуру слід розуміти не як аудіофайл (готовий звук), а як «креслення» вокалу: набір структурованих даних, з якого синтезатор щоразу заново генерує звучання.

Типова партитура зберігає кілька груп даних. Основу становлять ноти, кожна з яких має позицію в часі, тривалість та висоту (тон), і призначена кожній

ноті лірика – текстовий склад, який буде проспівано, а за потреби й точне фонетичне позначення його вимови (фонеми). Окремо зберігаються загальні музичні характеристики твору – темп (кількість ударів на хвилину, BPM) і тактовий розмір (наприклад, 4/4). Природність співу забезпечує крива висоти тону, що описує плавні відхилення висоти в межах та між нотами (портаменти, вібрато, інтонаційні переходи), а також інші параметри автоматизації – гучність, дихання, тембр тощо.

Саме партитура є об'єктом, з яким працює розроблюваний програмний продукт: він не синтезує звук, а перетворює партитури з одного формату в інший.

Кожен вокальний синтезатор зберігає партитури у власному форматі файлу: UTAU використовує формат UST, OpenUTAU – USTx, CeVIO – CCS, Synthesizer V – SVP, VOCALOID – VSQX та VPR. Ці формати відрізняються не лише розширенням, а й принципом організації даних: одні є простими текстовими файлами, інші – структурами XML, JSON чи YAML, ще інші запаковані в архів.

Через це партитура, створена в одному синтезаторі, не може бути напряду відкрита в іншому. На практиці музиканти й продюсери часто використовують кілька синтезаторів одночасно – для різних голосів, ефектів чи етапів роботи – і змушені переносити проекти між ними. За відсутності інструмента конвертації такий перенос виконується вручну: користувач наново вводить усі ноти, лірику та налаштування в цільовій програмі. Це трудомісткий процес, що забирає багато часу, призводить до помилок і відволікає від творчої роботи.

Отже, існує практична потреба в інструменті, що автоматично та точно переносить партитури між форматами зі збереженням усіх музичних параметрів.

З погляду інженерії програмного забезпечення розроблюваний продукт є системою перетворення (конвертації) файлів партитур вокальних синтезаторів, роботу якої доцільно формалізувати через входи, виходи та обмеження. Такий підхід дозволяє чітко окреслити межі системи ще до етапу проєктування та відокремити те, що вона повинна робити, від того, як саме це буде реалізовано. На вхід система отримує один або декілька завантажених користувачем файлів

партитур у підтримуваному форматі (UST, USTx, CCS, SVP, VPR або VSQX), обраний цільовий формат, а також специфічні для обраних форматів параметри конвертації – наприклад, кодування тексту вхідного файлу – та необов'язкові налаштування проміжних перетворень, зокрема транспонування ноти, заміну лірики й конвертацію вимови. Результатом її роботи є файл або набір файлів партитури у цільовому форматі, готовий до завантаження, що супроводжується переліком попереджень про можливу часткову втрату даних через структурні відмінності між форматами, а у разі некоректних чи не підтримуваних вхідних даних – повідомленням про помилку. Окремо слід наголосити, що попередження не перериває конвертацію, а лише інформує користувача про неточності, тоді як повідомлення про помилку означає, що результат сформувати неможливо.

Функціонально система має коректно переносити між форматами ноти, темп, тактовий розмір, лірику та криву висоти тону, підтримувати пакетну обробку кількох файлів і ізолювати збій конвертації одного файлу від решти. Водночас до неї висувається низка нефункціональних обмежень – простота розширення на нові формати, точність конвертації, зручність використання та незалежність від платформи користувача. Наведений поділ на функціональні та нефункціональні характеристики є відправною точкою для подальшого проєктування. Таким чином, з інженерного погляду задача зводиться до проєктування конвеєра «вхідний формат → уніфіковане проміжне представлення → вихідний формат», який зберігає максимум музичної інформації та явно сигналізує про неминучі втрати.

1.2. Аналіз аналогів програмного продукту

Перед проєктуванням власного рішення доцільно проаналізувати наявні програмні продукти, що розв'язують подібну задачу. Це дозволяє врахувати їх переваги та уникнути виявлених недоліків. Розглянемо три найпоширеніші рішення.

UtaFormatix – це веб-орієнтований конвертер партитур, що працює безпосередньо у браузері та не потребує встановлення додаткового програмного забезпечення (рис. 1.1) [10].

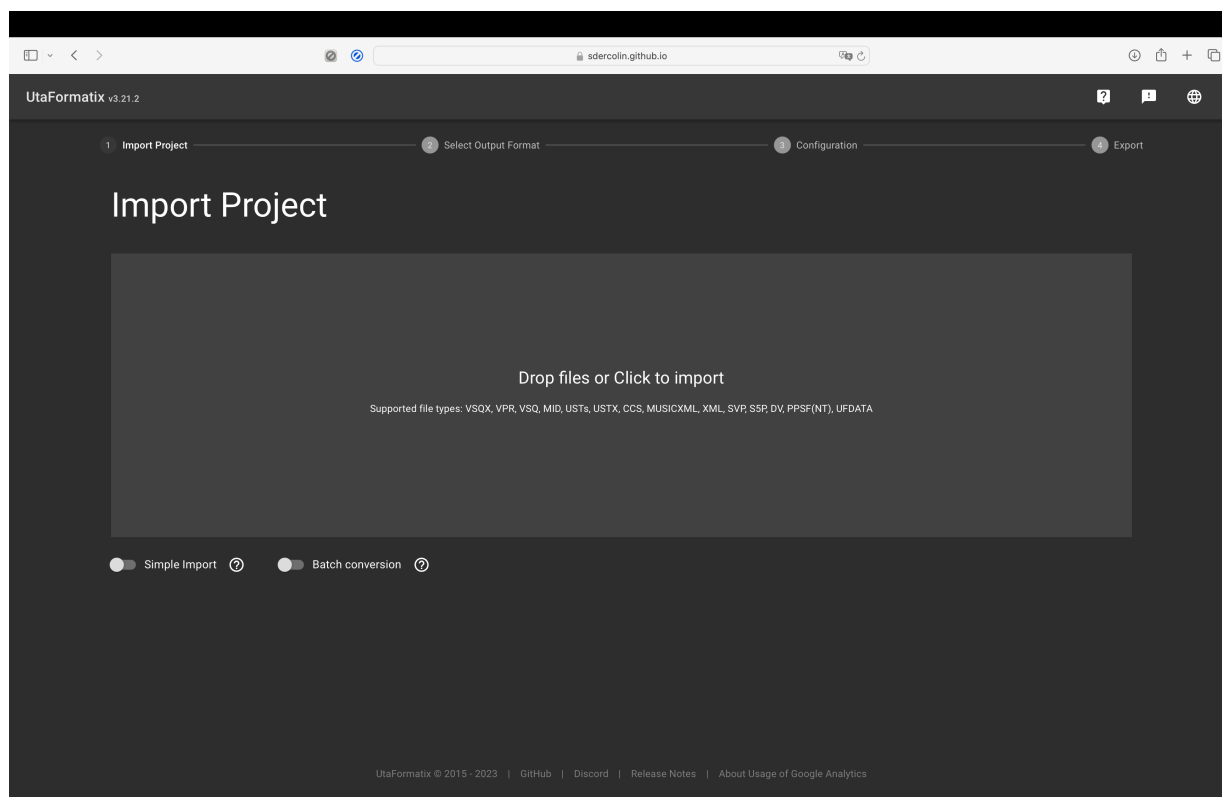


Рис. 1.1. Інтерфейс UtaFormatix

Основний функціонал програми побудовано навколо простого сценарію: користувач завантажує файл партитури, обирає цільовий формат зі списку та отримує готовий результат. Інструмент підтримує широкий перелік форматів вокальних синтезаторів і автоматизує весь процес конвертації, завдяки чому залишається зручним навіть для початківців, які не мають технічної підготовки. Окрім власне конвертації, UtaFormatix зберігає основні музичні параметри партитури – ноти, лірику та криву висоти тону. Водночас рішення має суттєві обмеження. Архітектурно воно побудоване так, що додавання підтримки нового формату є складним і трудомістким процесом, який потребує значних зусиль розробників. Крім того, у певних випадках конвертація передбачає перехід від відносної кривої висоти тону до абсолютної – наприклад, при перетворенні партитури SVP у формат CCS. Оскільки алгоритм не враховує особливостей

плавних переходів між нотами, крива висоти тону спотворюється, через що результат потребує ручного доопрацювання.

LibreSVIP – це універсальний конвертер проєктів вокальних синтезаторів, що, на відміну від UtaFormatix, встановлюється локально на комп'ютер користувача (рис. 1.2).

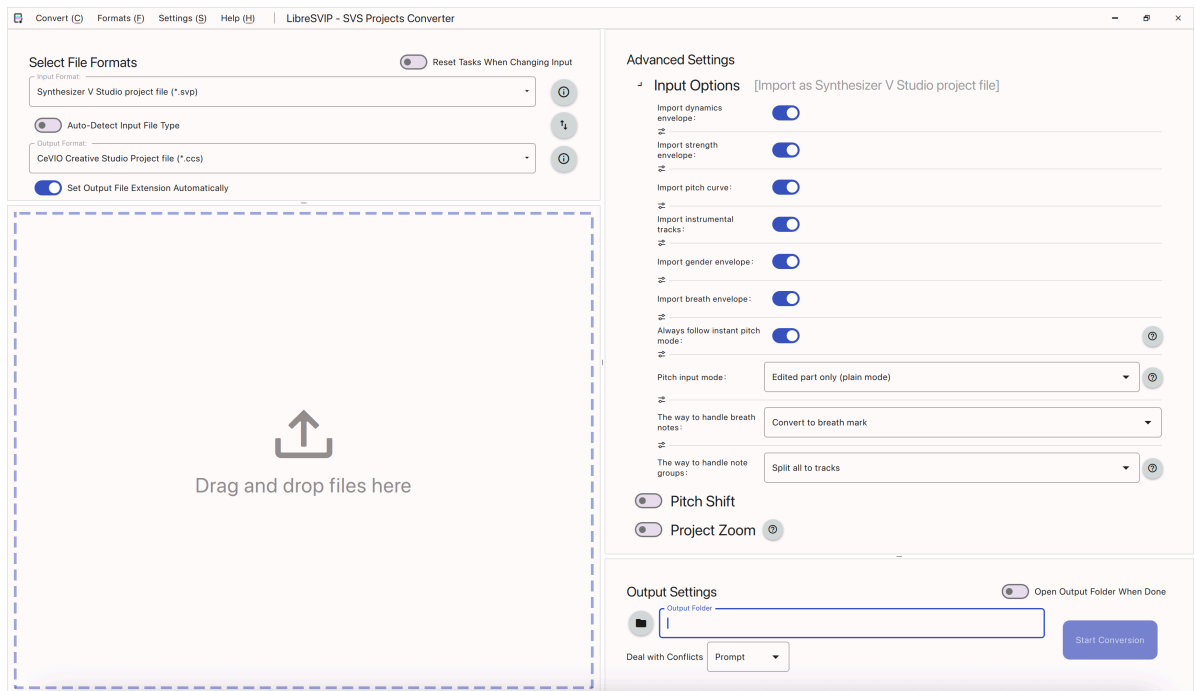


Рис. 1.2. Інтерфейс LibreSVIP

Сильною стороною цього рішення є підтримка великої кількості форматів партитур та перенесення значного обсягу музичних параметрів – програма прагне зберегти не лише ноти й лірику, а й детальні параметри автоматизації. Проте потреба у локальному встановленні сама собою знижує доступність інструмента порівняно з веб-рішеннями, а за відгукami користувачів програма працює нестабільно на окремих операційних системах, що ускладнює її використання для широкої аудиторії. Крім того, конвертація деяких форматів реалізована некоректно: зокрема, при перетворенні партитур формату SVP в інші формати виникають помітні артефакти кривої висоти тону, що знижує якість перенесення інтонації.

Окремий підхід демонструють самі вокальні синтезатори, частина з яких підтримує імпорт сторонніх форматів безпосередньо у власному редакторі.

Зокрема, Synthesizer V Studio дозволяє відкривати файли форматів UST та VSQX без використання окремого інструмента конвертації (рис. 1.3) [5].

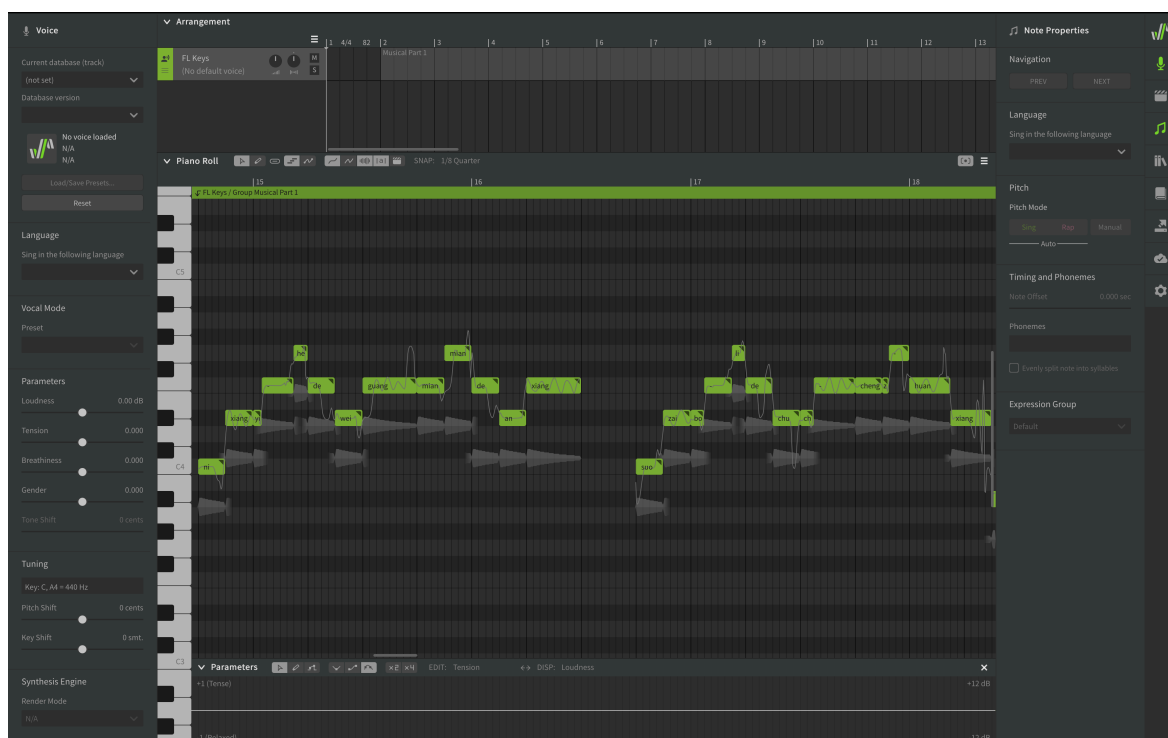


Рис. 1.3. Імпорт стороннього формату у Synthesizer V Studio

Перевагою такого підходу є зручність: конвертація вбудована в редактор, тому користувачу не потрібне стороннє програмне забезпечення, а імпортована партитура одразу готова до подальшого редагування. Однак ця можливість має подвійне обмеження. По-перше, вона замкнена на екосистему Synthesizer V: імпортувати дані можна лише в сам редактор, але не використати його як універсальний міст між іншими синтезаторами. По-друге, під час імпорту переносяться не всі параметри – зокрема, крива висоти тону з формату UST не імпортується, через що втрачаються інтонаційні нюанси оригінальної партитури.

Узагальнене порівняння розглянутих рішень із розроблюваним застосунком наведено в додатку А (таблиця А.1).

За результатами порівняння видно, що жодне з наявних рішень не поєднує одночасно веб-доступність без встановлення, коректне збереження кривої висоти тону між форматами з різним способом її зберігання та наявність проміжних перетворень. Виявлені недоліки аналогів – складність розширення,

спотворення пітч-кривої, прив'язка до однієї екосистеми, потреба у встановленні – стали орієнтирами при формулюванні вимог до власного продукту.

1.3. Аналіз форматів файлів партитур

Як було показано вище, файли партитур є водночас вхідними та вихідними даними розроблюваної системи: саме їх вона зчитує на вході та формує на виході. Тому коректне проєктування конвертера неможливе без детального розуміння внутрішньої будови кожного з форматів – того, як саме в них організовано та закодовано музичну інформацію. Без такого аналізу неможливо ані визначити, які параметри підлягають перенесенню, ані виявити структурні відмінності, що призводять до часткової втрати даних під час конвертації.

Нижче проаналізовано шість форматів, які підтримує розроблюваний застосунок: VSQX, VPR, UST, USTx, CCS та SVP. Для кожного з них визначено тип контейнера, спосіб організації даних, а також ключові структурні елементи, у яких зберігаються музичні параметри: ноти, темп, тактовий розмір, лірика, фонемі та крива висоти тону. Особливу увагу приділено тому, як кожен формат представляє висоту нот та криву висоти тону, оскільки саме ці аспекти найбільше різняться між форматами і є найскладнішими для коректного перенесення.

VSQX – формат файлів партитур вокального синтезатора VOCALOID версій 3–4 [4]. Дані зберігаються у вигляді XML-документа, що дозволяє структуровано організовувати інформацію про музичний твір.

Загальні параметри композиції (рис. 1.4.) – темп і тактовий розмір – зберігаються в елементі masterTrack: тактовий розмір задається елементом timeSig (атрибути m – міра, nu – чисельник, de – знаменник), а темп – елементом tempo (t – час встановлення, v – значення). Власне вокальні дані містяться в доріжках vsTrack, кожна з яких складається з блоків vsPart (вокальних груп із часом початку t, довжиною playTime та назвою name). Окрема нота описується елементом note, що зберігає лірику (y), фонемі (p), час початку (t), тривалість (dur) та MIDI-значення висоти (n), а точки автоматизації параметрів задаються елементами cc з позицією t і значенням v.

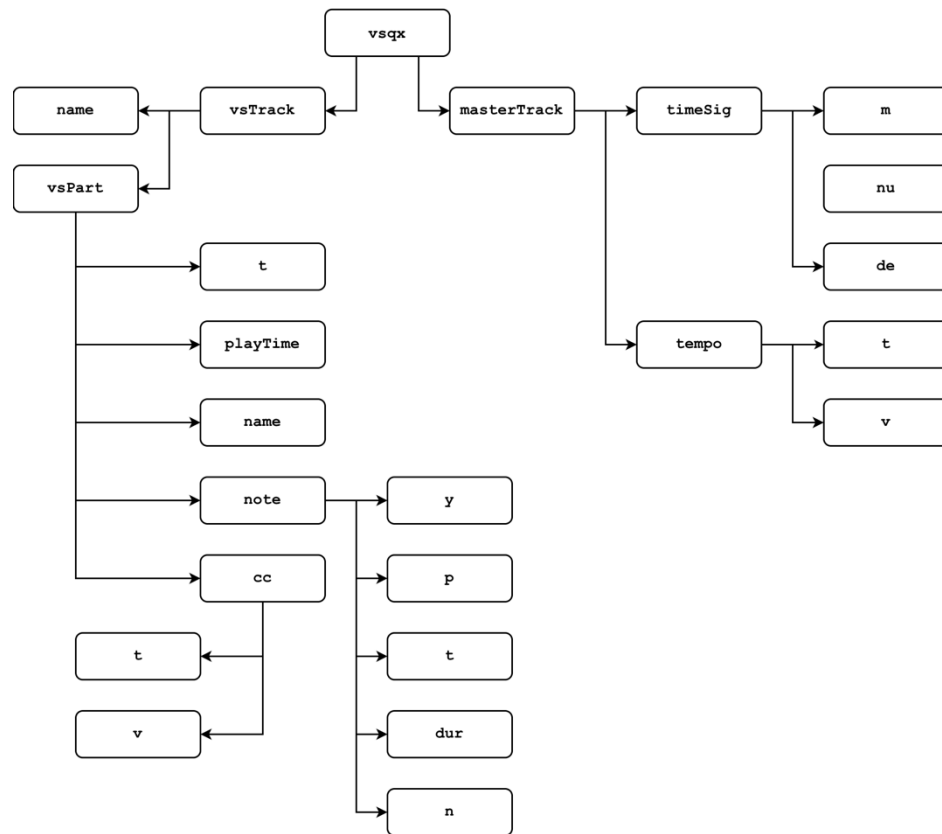


Рис. 1.4. Структура файлу VSQx

VPR – формат проєктів новіших версій VOCALOID. На відміну від VSQX, дані зберігаються у форматі JSON, запакованому в архів.

Структура VPR (рис. 1.5) подібна до VSQX, але організована інакше. На відміну від VSQX, дані зберігаються не як XML-документ, а у форматі JSON, запакованому в архів, що відображає загальну тенденцію новіших версій VOCALOID до переходу на легші для машинної обробки текстові формати. Загальні параметри композиції зберігає masterTrack, де темп (tempo) і тактовий розмір (timeSignature) задаються парами «позиція (pos) – значення (value)». Доріжки tracks містять об'єкти parts – вокальні групи з позицією pos і тривалістю duration, кожна з яких об'єднує масив нот notes та масив автоматизацій controllers. Кожна нота описується лірикою (lyrics), фонемами (phonemes), позицією (pos), тривалістю (duration) та MIDI-висотою (number), а кожен контролер автоматизації – назвою та масивом подій events із полями pos і value. Характерною рисою формату є те, що позиції тут задаються абсолютно – у тіках від початку композиції, завдяки чому кожна нота й кожна точка

автоматизації мають однозначну часову прив'язку незалежно від решти партитури.

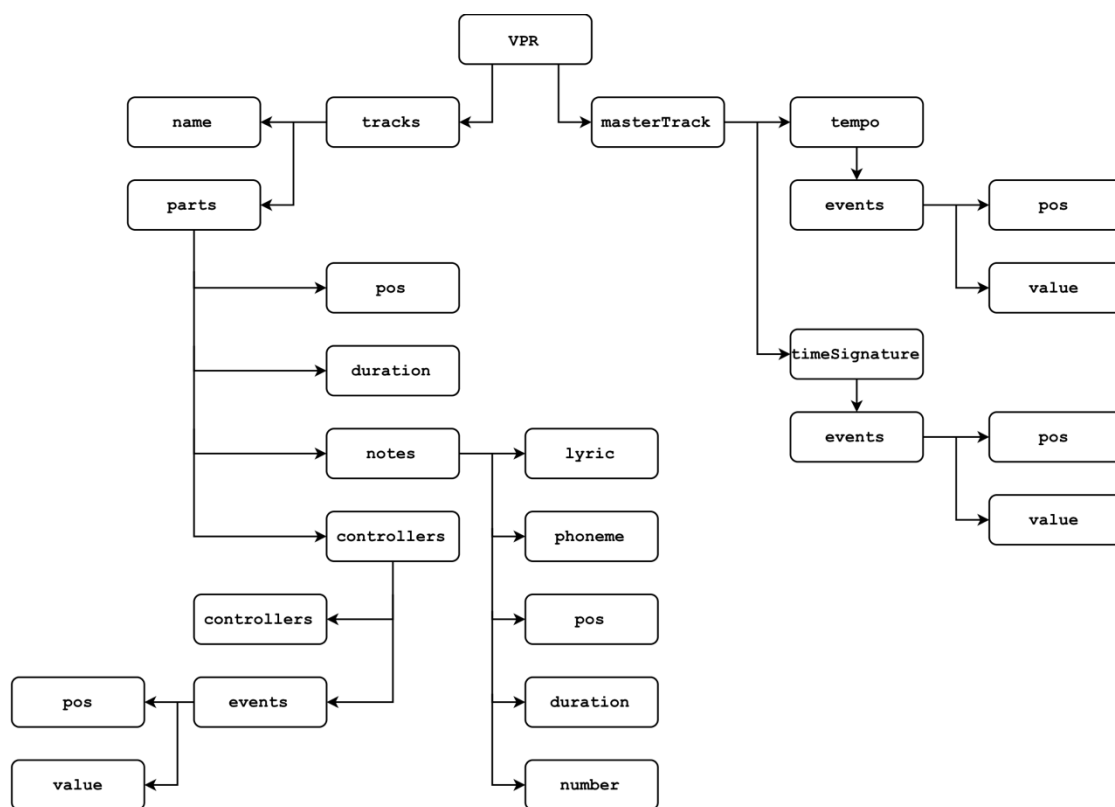


Рис. 1.5. Структура файлу VPR

UST – формат файлів партитур вокального синтезатора UTAU [7]. Особливістю UTAU є те, що він не підтримує окремі доріжки чи групи, тому одна партитура зберігає лише одну вокальну доріжку.

Файл формату UST (рис. 1.6.) складається з послідовності текстових секцій, кожна з яких виділена власним заголовком у квадратних дужках і описує або налаштування проєкту, або окрему ноту. Відкриває партитуру блок [#SETTINGS], що зберігає загальні параметри композиції (темп Tempo та назву проєкту ProjectName), за ним ідуть блоки окремих нот виду [#0000], де 0000 – порядковий номер ноти. Кожен такий блок містить тривалість (Length), MIDI-висоту (NoteNum), лірику (Lyric) та криву висоти тону у вигляді масиву точок (PitchBend), а завершує партитуру службовий блок [#TRACKEND]. Завдяки такій простій секційній організації файл UST залишається повністю текстовим і легко читається навіть без спеціального редактора, проте саме ця простота обмежує обсяг музичних даних, які формат здатен зберегти.

Дві важливі особливості формату:

1. Ноти не мають абсолютних позицій; позиція ноти обчислюється як сума тривалостей усіх попередніх нот.
2. Якщо між нотами є проміжок, на його місце вставляється спеціальна нота з лірикою «R» (англ. rest – пауза).

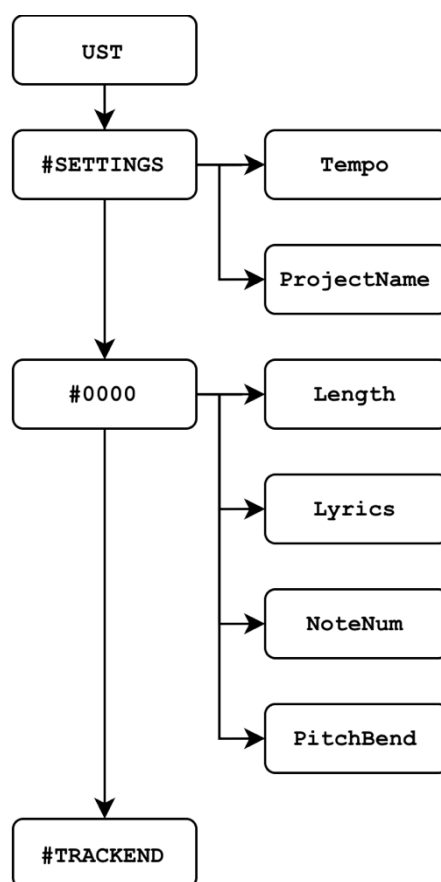


Рис. 1.6. Структура файлу UST

USTx – формат файлів проєктів OpenUTAU, сучасної відкритої версії UTAU [8]. Файл зберігається у форматі YAML, який є легко читабельним для людини. На відміну від UST, USTx підтримує кілька доріжок одночасно.

Файл формату USTx (рис. 1.7.) складається з ієрархічно впорядкованих YAML-блоків, що описують як загальні параметри композиції, так і вміст окремих доріжок, і завдяки текстовій природі формату легко читається людиною. Тактовий розмір зберігається у time_signatures (bar_position, beat_per_bar, beat_unit), а темп – у tempos (position, bpm). Доріжки tracks містять вокальні групи voice_parts із назвою (name), позицією (position) та тривалістю

(duration). Кожна нота описується позицією (position), тривалістю (duration), MIDI-висотою (tone) та лірикою (lyric), а криві автоматизації параметрів (curves) зберігаються як окремі масиви позицій (xs) і значень (ys) з назвою параметра (abbr). На відміну від простого секційного UST, така організація дозволяє зберігати кілька доріжок одночасно та чітко відокремлювати музичні дані від параметрів автоматизації.

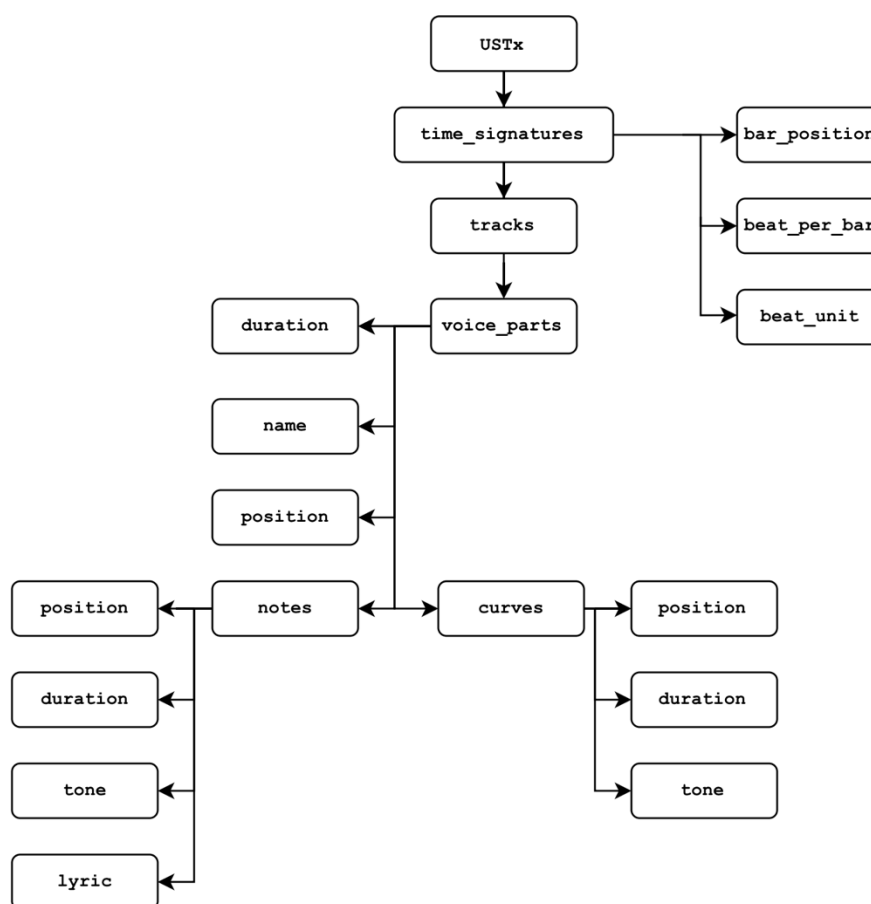


Рис. 1.7. Структура файлу USTx

CCS – формат партитур CeVIO. Файл формату CCS складається з XML-структури, у якій усі музичні дані згруповано всередині головного блоку Song. Цей блок містить кілька підблоків (рис. 1.8) – темп композиції (Tempo), тактовий розмір (Beat) та масив нот Score. Кожна нота Note описується атрибутами Clock (позиція старту), Duration (тривалість), Lyric (лірика) та Phonetic (фонемі), а її висота задається парою PitchOctave (номер октави) і PitchStep (номер ноти в октаві). Криву висоти тону зберігає окремий блок Parameters у вигляді послідовності точок LogF0. Розмітка XML робить файл самоописовим, оскільки призначення кожного значення зрозуміле вже з імені відповідного елемента.

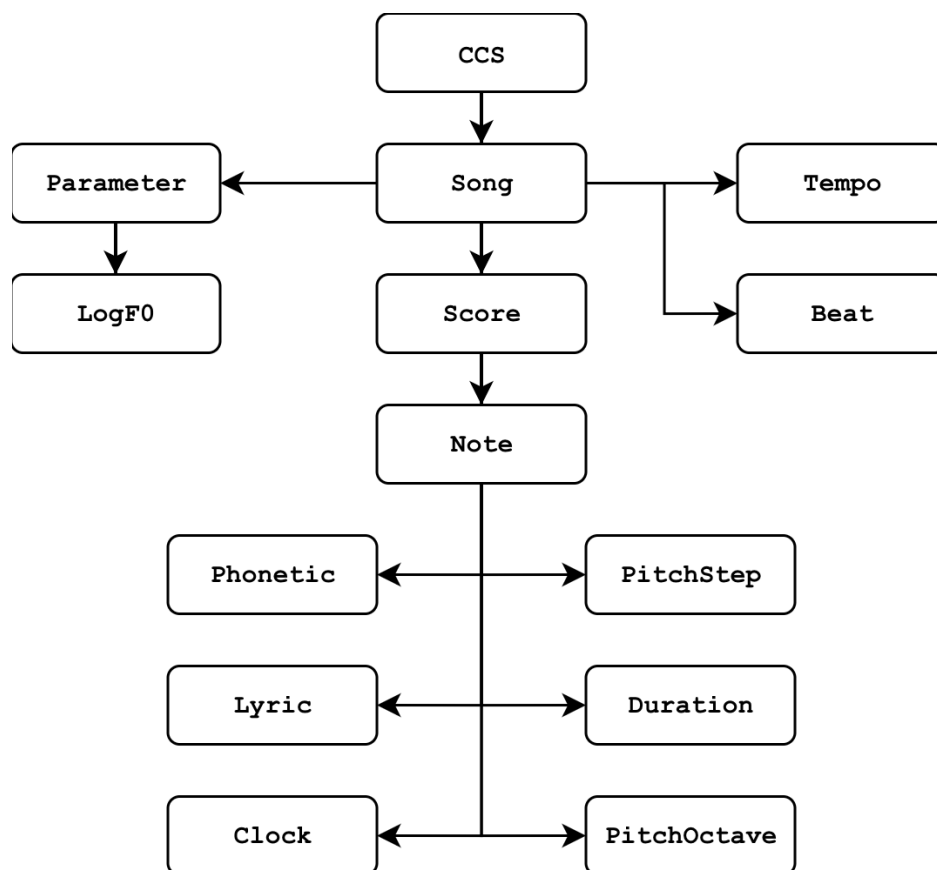


Рис. 1.8. Структура файлу CCS

SVP – формат партитур Synthesizer V Studio [9]. Файл формату SVP складається з єдиного JSON-документа (рис. 1.9.), організованого як набір вкладених об'єктів, що описують композицію, її доріжки та параметри. Об'єкт time зберігає темп і тактовий розмір композиції, а масив доріжок tracks – їх імена (name) та головну групу (mainGroup), що містить масив нот і об'єкт параметрів. Кожна нота описується позицією початку (onset), тривалістю (duration), лірикою (lyrics), фонемами (phonemes) та MIDI-висотою (pitch). Параметри висоти тону зберігаються в об'єкті parameters, зокрема масив точок кривої pitchDelta. Така деревоподібна організація даних робить формат гнучким і легко розширюваним, проте вимагає точного дотримання структури вкладеності при читанні та записі.

Особливістю SVP є те, що крива висоти тону зберігається у відносному вигляді (відхилення відносно базової кривої синтезатора), а сам синтезатор додає плавні переходи між нотами (портаменти). Саме неврахування цих переходів спричиняє згаданий раніше дефект у деяких аналогах.

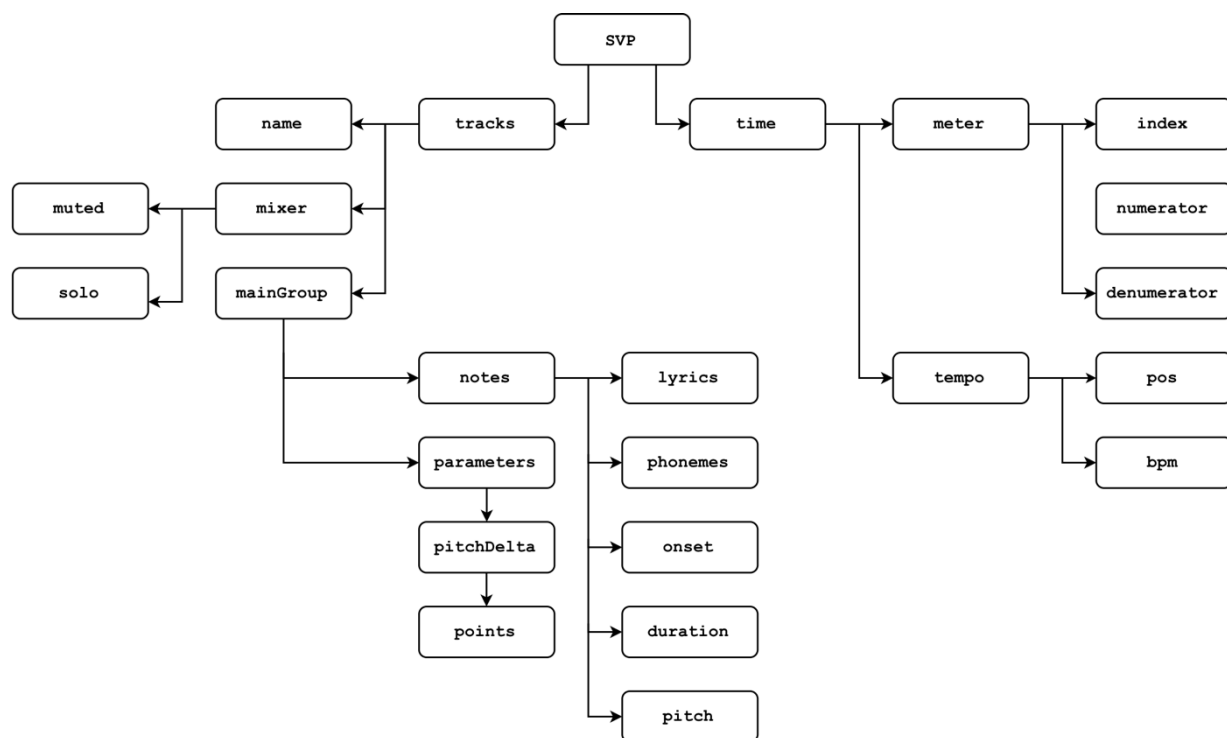


Рис. 1.9. Структура файлу SVP

Проведений аналіз показує, що формати суттєво відрізняються за способом зберігання даних: текстові (UST), XML (VSQX, CCS; пор. MusicXML [25]), JSON (SVP, VPR у архіві), YAML (USTx); за способом позиціонування нот (відносно у UST, абсолютне в інших); за представленням висоти (MIDI-номер, пара «октава– ступінь» у CCS); та за способом зберігання кривої висоти тону (абсолютна, логарифмічна, відносна). Ці відмінності формують ключові вимоги до проміжної моделі даних, яка має узагальнити всі формати, та обґрунтовують потребу в єдиному уніфікованому представленні.

1.4. Обґрунтування вибору архітектурного підходу та інструментальних засобів

На основі поставленої задачі та проведеного аналізу аналогів обґрунтуємо вибір архітектурного підходу й інструментальних засобів реалізації розроблюваного продукту. Вибір архітектури визначає не лише технічну реалізацію, а й безпосередньо впливає на доступність застосунку для кінцевого користувача, рівень захищеності його даних та витрати на подальше супроводження системи, тому до цього питання слід підходити системно.

Аналіз аналогів виявив, що десктопні рішення (як-от LibreSVIP) мають суттєвий недолік – потребу у встановленні та залежність від операційної системи користувача. Такі застосунки вимагають окремих збірок під різні платформи, періодичного оновлення, а також наявності в користувача прав на встановлення стороннього програмного забезпечення, що часто є проблемою в корпоративному чи навчальному середовищі з обмеженими правами доступу. Окрім того, десктопний дистрибутив прив'язаний до конкретної версії операційної системи та її бібліотек, через що з часом виникають проблеми сумісності. Усе це знижує доступність і стабільність рішення. Натомість веб-орієнтований підхід дозволяє запускати застосунок у будь-якому сучасному браузері без встановлення, що забезпечує кросплатформність та простоту доступу: користувачеві достатньо відкрити сторінку за посиланням, аби одразу почати роботу, незалежно від того, яким пристроєм чи операційною системою він користується.

Водночас класичний веб-підхід із серверною обробкою має власні недоліки. По-перше, він потребує постійного утримання серверної інфраструктури, що пов'язано з фінансовими витратами та необхідністю адміністрування. По-друге, постає питання масштабування під навантаженням: за одночасного звернення великої кількості користувачів сервер може стати «вузьким місцем» системи, а обробка ресурсомістких файлів на стороні сервера лише посилює цю проблему. По-третє, і це особливо важливо, виникає питання конфіденційності – файли користувача доводиться передавати на сервер, де вони тимчасово зберігаються та обробляються, що створює потенційні ризики витоку даних і вимагає додаткових заходів із їх захисту. Тому для розроблюваного продукту обрано підхід клієнтського односторінкового застосунку (Single Page Application), у якому вся обробка файлів виконується безпосередньо у браузері користувача, без передавання даних на сервер. За такого підходу файл ніколи не залишає пристрій користувача, що повністю усуває ризики, пов'язані з його передаванням мережею. Це поєднує переваги веб-доступності з

конфіденційністю, відсутністю серверних витрат та незалежністю від стабільності мережевого з'єднання після завантаження сторінки.

Оскільки обробка виконується у браузері, основним середовищем виконання є JavaScript – єдина мова, що нативно підтримується усіма сучасними браузерами. Однак безпосереднє використання JavaScript для системи з нетривіальною логікою має обмеження, передусім через динамічну типізацію, за якої багато помилок виявляються лише під час виконання. Тому як основну мову розробки обрано TypeScript – статично типізовану надмножину JavaScript, яка компілюється у звичайний JavaScript і зберігає повну сумісність з ним [30]. Оскільки система оперує складними вкладеними структурами даних (партитури → доріжки → ноти → параметри), статична типізація є критично важливою: вона дозволяє виявляти помилки невідповідності типів ще на етапі компіляції, а не під час виконання. Це особливо суттєво для задачі конвертації форматів, де кожен формат має власну структуру, і помилка в зіставленні полів між форматами може призвести до спотворення музичних даних. Окрім раннього виявлення помилок, статична типізація слугує формою документації коду, спрощує рефакторинг та підтримку проєкту і покращує підказки в середовищі розробки, що в сукупності підвищує надійність і керованість кодової бази.

Для побудови інтерфейсу обрано бібліотеку React та метафреймворк Next.js, що надають реактивну компонентну модель та інфраструктуру збирання застосунку [13] [14]. Компонентна модель React дозволяє декомпонувати інтерфейс на незалежні повторно використовувані частини, а реактивний підхід забезпечує автоматичне оновлення відображення у відповідь на зміну стану, що спрощує побудову інтерактивного інтерфейсу. Next.js, своєю чергою, надає готову інфраструктуру збирання, маршрутизації та оптимізації застосунку, позбавляючи від необхідності налаштовувати ці механізми вручну, а також дозволяє згенерувати статичну версію застосунку, яку можна розмістити без серверної частини – що повністю узгоджується з обраною клієнтською архітектурою. Для стилізації обрано утилітарний CSS-фреймворк Tailwind CSS, який дає змогу швидко описувати оформлення безпосередньо в розмітці за

допомогою набору утилітарних класів, забезпечуючи узгодженість візуального стилю та зменшуючи обсяг власних CSS-стилів [15].

Оскільки застосунок є клієнтським веб-рішенням, а вся обробка виконується наявними засобами браузера, вимоги до обладнання користувача є мінімальними. Продукт не виконує ресурсомістких фонових обчислень, не встановлює додаткових компонентів у систему та не залежить від продуктивності серверної інфраструктури, тому здатний працювати навіть на застарілих або не потужних пристроях. Узагальнено вимоги наведено в табл.1.1.

Таблиця 1.1.

Вимоги до апаратного та програмного забезпечення

Компонент	Мінімальні вимоги
Процесор	будь-який сучасний x86-64 або ARM процесор
Оперативна пам'ять	2 ГБ і більше
Дискове сховище	не потребує встановлення; лише місце для збереження результатів
Програмне забезпечення	сучасний браузер (Chrome, Firefox, Safari, Edge)
Операційна система	будь-яка (Windows, macOS, Linux), що підтримує сучасний браузер
Мережа	потрібна лише для початкового завантаження сторінки

Як видно з таблиці, продукт не висуває спеціальних вимог до обладнання та не потребує встановлення. Єдиною суттєвою умовою його роботи є наявність сучасного браузера, який сьогодні присутній фактично на будь-якому пристрої, а мережеве з'єднання потрібне лише одноразово – для завантаження сторінки, після чого застосунок здатний працювати автономно. Це повністю відповідає поставленій вимозі незалежності від платформи та забезпечує максимально широку доступність продукту для кінцевих користувачів.

Висновки до першого розділу

У першому розділі проведено аналіз предметної області та сформульовано задачу розробки системи обробки партитур вокальних синтезаторів.

У підрозділі 1.1 розкрито поняття вокального синтезатора як засобу штучного відтворення співу та поняття партитури як проєктного файлу, що зберігає ноти, лірику, фонему, темп, тактовий розмір та криву висоти тону. Виявлено ключову проблему предметної області – несумісність форматів партитур різних синтезаторів, що змушує користувачів переносити проєкти вручну. З погляду інженерії програмного забезпечення задачу формалізовано через входи (файли партитур, цільовий формат, параметри) та виходи (конвертований файл, попередження, повідомлення про помилки), звівши її до проєктування конвеєра «вхідний формат → уніфіковане представлення → вихідний формат».

У підрозділі 1.2 проаналізовано три аналоги – UtaFormatix, LibreSVIP та вбудовану конвертацію Synthesizer V – і за допомогою порівняльної таблиці виявлено їх спільні недоліки: складність розширення на нові формати, спотворення кривої висоти тону, прив'язку до однієї екосистеми та потребу у встановленні. Ці недоліки стали орієнтирами при формулюванні вимог до власного продукту.

У підрозділі 1.3 детально проаналізовано структуру шести форматів файлів партитур (VSQX, VPR, UST, USTx, CCS та SVP), визначено їх типи контейнерів та принципові відмінності у способах зберігання нот, висоти та кривої висоти тону. Ці відмінності формують вимоги до уніфікованої проміжної моделі даних.

У підрозділі 1.4 обґрунтовано вибір клієнтського веб-підходу (SPA) та інструментальних засобів (TypeScript, React, Next.js, Tailwind CSS), а також визначено мінімальні вимоги до апаратного та програмного забезпечення. Детальне проєктування архітектури системи на основі зроблених висновків наведено в наступному розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СТРУКТУРИ ДАНИХ ТА АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Дизайн архітектури та проєктування системи

Система організована як односторінковий вебдодаток, що функціонує повністю у браузері користувача без необхідності завантаження файлів на сервер або встановлення будь-якого додаткового програмного забезпечення. Принципова відмінність цієї архітектури від серверно-орієнтованих підходів полягає в тому, що вся логіка обробки партитур – парсинг, конвертація, серіалізація – виконується безпосередньо на стороні клієнта в середовищі виконання JavaScript (рушій V8, вбудований у браузер). Це означає, що файли партитур жодного разу не залишають пристрій користувача, що є суттєвою перевагою з точки зору конфіденційності та незалежності від з'єднання з мережею після завантаження сторінки. Крім того, перенесення всіх обчислень на бік клієнта знімає з розробника відповідальність за збереження користувацьких даних і дозволяє розгорнути застосунок як набір статичних файлів, що суттєво спрощує його супровід.

Архітектурно систему можна розділити на чотири логічні шари, що взаємодіють між собою (рис. 2.1). Шар представлення утворюють React-компоненти, які формують інтерфейс – зону завантаження файлів, панелі вибору формату з параметрами та список результатів конвертації; вони реалізовані на TypeScript з використанням React 19 і стилізовані за допомогою Tailwind CSS 4. Шар стану відповідає за реактивний стан застосунку, яким керують хуки React (useState, useCallback, useMemo): сюди входять список доданих файлів, поточно обрані вхідний і вихідний формати, значення параметрів кожного конвертера, а також карта результатів і статусів конвертації. Шар бізнес-логіки об'єднує функції конвертації, реєстр плагінів і канонічну модель даних, а шар плагінів складається з набору незалежних модулів, кожен з яких реалізує читання та запис одного конкретного формату партитур. Такий чіткий розподіл відповідальностей

між шарами знижує зв'язаність компонентів і дозволяє змінювати чи розширювати кожен шар окремо, не зачіпаючи решту системи.

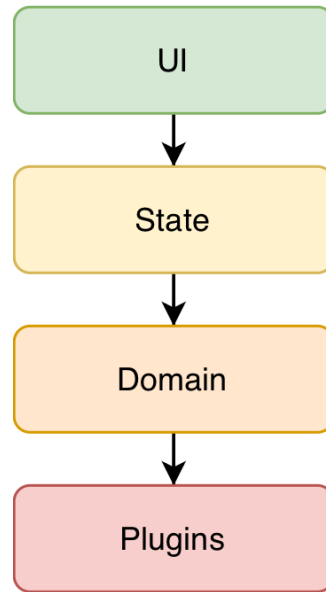


Рис. 2.1. Шари архітектури системи

Для реалізації проєкту було обрано TypeScript як основну мову розробки [11]. TypeScript є статично типізованим надмножиною JavaScript, що компілюється у звичайний JavaScript, – це дозволяє виявляти клас помилок, пов'язаних із типами, на етапі компіляції, а не під час виконання. Оскільки система оперує складними вкладеними структурами даних (партитури → доріжки → ноти → параметри автоматизації), строга типізація є критично важливою: вона унеможливорює передачу об'єктів неправильного типу між компонентами системи та генерує зрозумілі повідомлення про помилки ще у редакторі коду. Вся кодова база, включаючи плагіни та модель даних, написана на TypeScript – це забезпечує єдність мовної бази та гарантує узгодженість інтерфейсів між модулями.

Фреймворк Next.js версії 16 обрано як основу для побудови SPA. Next.js є React-метафреймворком, що надає систему маршрутизації, оптимізацію зображень та шрифтів, підтримку серверних та клієнтських компонентів, а також інструменти для збирання та оптимізації пакету, що виконуються у середовищі Node.js [13] [16]. Для даного проєкту особливо важливою є підтримка директиви "use client", що дозволяє явно позначати компоненти, які мають виконуватися у

браузері – це необхідно для роботи з File API , Blob API та іншими браузерними інтерфейсами, що використовуються при обробці файлів [17] [18].

React 19 забезпечує реактивну модель оновлення інтерфейсу, при зміні стану (наприклад, зміна списку файлів або обраного формату) React автоматично перерендерує лише ті компоненти, які залежать від зміненого стану [14]. Це означає, що оновлення статусу конвертації одного файлу у списку результатів не призводить до перемальовування всього застосунку – лише відповідного елемента списку.

Tailwind CSS 4 обрано для стилізації як утилітарний CSS-фреймворк [15]. На відміну від компонентних бібліотек, Tailwind не нав'язує готових компонентів, а натомість надає набір атомарних CSS-класів, що застосовуються безпосередньо у JSX-розмітці. Це забезпечує максимальну гнучкість при розробці власного дизайну без необхідності перевизначати чужі стилі. Tailwind 4 підтримує темну тему через клас .dark на кореновому елементі, що дозволяє реалізувати перемикання теми без перезавантаження сторінки через збереження вибору у localStorage.

Важливою архітектурною характеристикою є відсутність серверних залежностей під час роботи. Після того, як браузер завантажив JavaScript-пакет застосунку, система може конвертувати файли без будь-якого підключення до мережі. Відмова від сервера суттєво спрощує розгортання (достатньо статичного хостингу), знімає питання масштабування під навантаженням та гарантує повну конфіденційність даних користувача.

Слід окремо наголосити на специфіці взаємодії компонентів у такій архітектурі. Хоча система є вебдодатком, вона не використовує класичну клієнт-серверну модель із обміном даними через HTTP/REST або WebSocket. Сервер задіяний виключно для початкової віддачі статичних файлів (HTML, JavaScript, CSS) і не бере участі в обробці партитур. Уся взаємодія відбувається в межах одного середовища виконання браузера: шар інтерфейсу (React-компоненти) звертається до ядра конвертації безпосереднім викликом функції convertFile, передаючи їй вміст файлу у вигляді масиву байтів та параметри, і синхронно

отримує результат. Таким чином, межа «фронтенд – ядро» є межею між модулями всередині того самого процесу, а не мережевою межею між клієнтом і сервером. Відсутність мережевого обміну на цій межі є свідомим архітектурним рішенням, що впливає з вимоги конфіденційності, а не наслідком неповноти реалізації.

2.2. Функціональні та нефункціональні вимоги до системи

Сформульовані в попередніх підрозділах висновки аналізу предметної області та аналогів дозволяють перейти від загального розуміння задачі до переліку конкретних вимог, яким повинна відповідати система. Вимоги поділено на дві групи: функціональні, що визначають перелік операцій і поведінок, які система зобов'язана реалізовувати, та нефункціональні, що задають якісні характеристики й обмеження. Такий поділ є стандартним для інженерії програмного забезпечення й дозволяє відокремити те, що система робить, від того, наскільки якісно вона це робить.

Функціональні вимоги сформовано на основі аналізу предметної області та потреб цільової аудиторії – музикантів і продюсерів, що використовують вокальні синтезатори. Передусім система повинна підтримувати читання та запис файлів у шести форматах партитур – UST (UTAU), USTx (OpenUTAU), CCS (CeVIO), SVP (Synthesizer V Studio), а також VPR і VSQX (VOCALOID). При цьому кожен формат реалізується у вигляді окремого плагіну з уніфікованим інтерфейсом, що забезпечує ізолюваність логіки обробки та незалежність плагінів один від одного: збій або зміна в одному плагіні не повинні впливати на решту. Під час перетворення між форматами система має коректно переносити ключові музичні параметри – усі ноти партитури (з позицією початку, тривалістю, MIDI-номером та лірикою), позначки темпу й тактового розміру, а також криву висоти тону, оскільки саме цей набір є мінімально необхідним для відтворення вокальної партії в цільовому синтезаторі. Перенесення кривої висоти тону винесено в окрему вимогу свідомо, адже саме її некоректна обробка є основним недоліком наявних аналогів.

Окрему групу функціональних вимог становить підтримка пакетної обробки. Для підвищення продуктивності роботи система повинна давати змогу додавати кілька файлів за один раз – як перетягуванням, так і через стандартний діалог вибору файлів – і обробляти кожен файл незалежно, відображаючи результати у списку з індивідуальним статусом і кнопкою завантаження. За наявності кількох успішних результатів система має надавати можливість завантажити їх усі одночасно. Принциповою є вимога ізоляції збоїв: кожне завдання конвертації повинно мати чітко визначений стан (очікує, виконується, завершено успішно або з помилкою), а збій обробки одного файлу не має переривати обробку решти пакета. Окремо виділено механізм попереджень: у разі нефатальних ситуацій – наприклад, коли окремий параметр ноти втрачається через відсутність відповідного поля у цільовому форматі – система має відображати попередження з описом ситуації, що дозволяє користувачу свідомо оцінити якість результату, не перериваючи при цьому конвертацію. Принципова відмінність попередження від помилки полягає в тому, що попередження інформує про часткову неточність, тоді як помилка означає неможливість сформувати результат узагалі.

Окрім базової конвертації, система повинна надавати засоби налаштування та додаткової обробки. Для кожного формату можуть бути доступні параметри, що впливають на логіку читання або запису (наприклад, кодування текстового файлу – UTF-8 чи Shift-JIS), які відображаються у вигляді динамічно згенерованої форми на основі метаданих плагіну, без жорстко закодованих знань інтерфейсу про конкретні формати. Для зручності типового сценарію вхідний формат повинен визначатися автоматично за розширенням першого доданого файлу, аби в більшості випадків користувачу залишалось обрати лише цільовий формат. Нарешті, для підтримки багатомовних партитур система має забезпечувати конвертацію тексту лірики з ієрогліфічних систем письма у латинізовані форми – пін'їн для китайської та ромаджі для японської мови – у випадках, коли цього вимагає цільовий формат або синтезатор. Ці засоби додаткової обробки виносяться в окремий конвеєр проміжних перетворень.

Нефункціональні вимоги визначають якісні характеристики системи й значною мірою зумовлюють подальші архітектурні рішення. Ключовою серед них є клієнтська обробка без завантаження даних: уся обробка файлів повинна відбуватися виключно у браузері, без передавання їх вмісту на зовнішній сервер, що забезпечує як конфіденційність, так і незалежність від мережі. Не менш важливою є легка розширюваність архітектури – додавання підтримки нового формату повинне зводитися до реалізації єдиного інтерфейсу Converter та реєстрації плагіну, без змін у компонентах інтерфейсу, конвеєрі конвертації чи логіці реєстру. Із цією вимогою тісно пов'язана незалежність від платформи: оскільки система є вебдодатком, вона має коректно працювати у будь-якому сучасному браузері (Chrome, Firefox, Safari, Edge) на будь-якій операційній системі та не потребувати встановлення стороннього середовища виконання.

Окрему групу становлять вимоги до якості роботи. Система повинна забезпечувати високу швидкість – конвертація одного файлу має завершуватися практично миттєво на сучасному пристрої, для чого обчислювально інтенсивні операції оптимізуються так, щоб не блокувати відповідь інтерфейсу. Точність конвертації є критичною: усі ноти, темп, тактовий розмір та крива висоти тону повинні переноситися коректно, а неминучі через структурні відмінності форматів втрати – чітко документуватися у вигляді попереджень. Система також має коректно обробляти помилки, ізолюючи збій конвертації одного файлу від решти пакета та подаючи зрозуміле повідомлення безпосередньо під відповідним файлом. Нарешті, застосунок повинен підтримувати темний і світлий режими відображення з автоматичним визначенням системних налаштувань браузера та збереженням вибору користувача між сесіями, що підвищує зручність тривалої роботи.

Сформульовані функціональні та нефункціональні вимоги тісно пов'язані між собою і разом визначають характер подальших проєктних рішень. Так, вимога підтримки багатьох форматів у поєднанні з вимогою простоти розширення безпосередньо зумовлює потребу у плагінній архітектурі та єдиному проміжному представленні даних, а вимоги клієнтської обробки й незалежності

від платформи – вибір веб-орієнтованого стека, що виконується у браузері. Точність конвертації та коректне відображення попереджень, своєю чергою, висувають вимоги до самої моделі даних, яка повинна зберігати всі ключові музичні параметри без втрат. Таким чином, наведені вимоги є відправною точкою для проєктування канонічної моделі та архітектури системи, що детально розглядаються в наступних підрозділах.

2.3. Канонічна модель та конвеєр конвертації

Центральним елементом архітектури є канонічна проміжна модель – набір TypeScript-класів, що описують уніфіковане представлення вокальної партитури, незалежне від будь-якого конкретного формату файлу. Замість того щоб конвертувати дані напряму з одного формату в інший, система спочатку нормалізує вхідну партитуру у це проміжне представлення, а вже з нього формує файл цільового формату. Такий підхід відповідає мережевій моделі hub-and-spoke (рис. 2.2), у якій канонічна модель відіграє роль центрального вузла (hub), а кожен плагін формату – промені (spokes), що з'єднуються лише з ним: плагін перетворює дані зі свого формату у канонічне представлення (операція читання) або з канонічного представлення у свій формат (операція запису), але ніколи не взаємодіє з іншими плагінами напряму.

Перевага такого рішення стає очевидною при оцінці його масштабованості. Для підтримки N форматів достатньо реалізувати N плагінів (по одному парсеру та генератору на формат), тоді як при прямій конвертації між усіма парами форматів знадобилося б порядку $N \times (N-1)$ окремих перетворювачів. Окрім того, додавання кожного нового формату не зачіпає вже наявні плагіни, оскільки всі вони «спілкуються» виключно через спільну модель, що робить систему лінійно розширюваною та значно спрощує її підтримку.



Рис. 2.2. Hub-and-spoke модель між плагінами та канонічною моделлю

Канонічну модель утворює кілька класів, кожен з яких відповідає за окрему сутність партитури – від кореневого об'єкта до окремої ноти та кривих автоматизації. Їх призначення та ключові поля узагальнено в табл. 2.1.

Таблица 2.1.

Класи канонічної моделі та їх ключові поля

Клас	Призначення	Ключові поля
Score	Кореневий клас, що агрегує всі дані партитури	tempoList: SongTempo[], timeSignatureList: TimeSignature[], trackList: Track[]
SongTempo	Одна позначка темпу	position (тіки), bpm
TimeSignature	Одна позначка тактового розміру	barIndex, numerator, denominator; метод barLength()
SingingTrack	Вокальна доріжка (успадковує TrackMixer)	noteList: Note[], editedParams: Params

InstrumentalTrack	Інструментальна доріжка (успадковує TrackMixer)	audioFilePath, offset
Note	Окрема нота вокальної доріжки	startPos, length, keyNumber (MIDI), lyric, editedPhones, vibrato; обчислюване endPos
Params	Набір кривих автоматизації	pitch, volume, breath, gender, strength

Кореневий клас Score реалізований як звичайний TypeScript-клас із конструктором, що приймає Partial, це дозволяє ініціалізувати об'єкт лише тими полями, які потрібні у конкретному контексті, що особливо зручно при поетапному наповненні моделі під час парсингу. Доріжки в моделі представлені двома класами – SingingTrack та InstrumentalTrack, – обидва з яких успадковують абстрактний клас TrackMixer зі спільними полями мікшування (title, mute, solo, volume, pan) і розрізняються за полем type зі значенням "Singing" або "Instrumental". Завдяки цьому загальна логіка роботи з доріжками не дублюється, а специфічні для кожного типу поля додаються лише у відповідному підкласі.

Клас TimeSignature, окрім полів чисельника та знаменника, надає метод barLength(): number, що обчислює довжину одного такту у тіках за формулою:

$$(\text{ticksInBeat} \times 4 \times \text{numerator}) / \text{denominator}$$

Він використовується плагінами при переведенні позицій між тіками та номерами тактів. Клас Note, що є центральним об'єктом партитури, додатково надає обчислюване поле endPos, яке дорівнює startPos + length, це зручно при перевірці перекриттів або відстаней між нотами у плагінах.

Час у канонічній моделі відраховується у тіках. Одиниця вимірювання визначена константою ticksInBeat = 480, тобто одна чверть ноти при будь-якому темпі завжди дорівнює 480 тіків. Така нормалізація відповідає стандарту MIDI і

дозволяє зберігати позиції нот незалежно від темпу [24]. При зміні BPM позиції нот у тіках залишаються незмінними. Для перерахунку тіків у реальний час у секундах використовується формула:

$$t(c) = \text{position_in_ticks} / (\text{bpm} / 60 \times \text{ticksInBeat})$$

Вона застосовується в плагінах, що оперують абсолютними часовими мітками (наприклад, формат CCS).

Об'єкт Params агрегує п'ять кривих автоматизації – висоти тону (pitch, відхилення у сентах відносно нотної висоти), динаміки гучності (volume), аспірації (breath), висоти формант (gender) та сили голосу (strength). Кожна з цих кривих є екземпляром класу ParamCurve, який зберігає впорядкований масив точок Point, де кожна точка задається парою «позиція у тіках – значення». Таким чином, модель утворює послідовну ієрархію Score → SingingTrack/InstrumentalTrack → Note та Params → ParamCurve → Point, що повністю описує партитуру незалежно від конкретного формату.

Найскладнішим аспектом конвертації є коректне перенесення кривої висоти тону, оскільки різні формати зберігають її принципово по-різному: одні – як набір контрольних точок із певним типом інтерполяції між ними, інші – як щільно дискретизовану послідовність значень, ще інші – у вигляді відносних відхилень, що накладаються на власну «базову» криву синтезатора. Саме на цьому етапі найчастіше виникають спотворення, зокрема описаний у першому розділі дефект рішення UtaFormatix при переході SVP в CCS. Щоб уникнути цих спотворень, у канонічній моделі прийнято єдине представлення кривої висоти тону як послідовності точок (t, d), де t – позиція у тіках, а d – відхилення висоти у сентах відносно висоти відповідної ноти (1 напівтон = 100 сентів). Кожен плагін відповідає за приведення власної кривої до цього представлення при читанні та матеріалізацію з нього при записі.

Ключова відмінність від UtaFormatix полягає в обробці форматів із відносною кривою (зокрема SVP). У таких форматах фактична висота тону складається з власної базової кривої синтезатора, що містить плавні переходи між сусідніми нотами (портаменто), на яку накладаються користувацькі

відхилення. Якщо при конвертації в абсолютну криву враховувати лише відхилення й ігнорувати базову, переходи між нотами «ламаються» – саме цей дефект і має UtaFormatix. У розробленій системі для кожного формату з відносною кривою реалізовано симулятор базової висоти, що відтворює форму нотної огинаючої з урахуванням переходів. Для Synthesizer V симулятор моделює перехід між нотами сигмоїдною функцією, параметри якої визначаються тривалістю портаменто та зсувами, а поверх неї додає шари вібрато та користувацького контролю висоти. Перетворення відносної кривої в абсолютну виконується шляхом дискретизації симулятора з фіксованим інтервалом і додавання до отриманої огинаючої збережених відхилень. Завдяки цьому переходи між нотами зберігаються коректно навіть при конвертації у формати з абсолютною кривою, що усуває недолік, притаманний наявним аналогам.

2.4. Архітектура плагінів та реєстр конвертерів

Архітектуру підтримки форматів побудовано за патерном плагінної (мікроядерної) архітектури [29]. Згідно з цим патерном система поділяється на стабільне ядро з мінімальною фіксованою відповідальністю та набір незалежних розширень (плагінів), що підключаються до ядра через єдиний контракт. У розробленій системі роль ядра виконують канонічна модель Score і конвеєр конвертації, а роль розширень – плагіни форматів. Вибір цього патерну зумовлений основною нефункціональною вимогою – простотою додавання нових форматів: ядро не містить жодних знань про конкретні формати, тому додавання формату не потребує його модифікації. Завдяки цьому кожен плагін можна розробляти, тестувати й супроводжувати ізольовано, не зачіпаючи решту системи, що знижує ризик внесення помилок під час розширення функціональності. Така ізоляція також спрощує паралельну роботу кількох розробників, оскільки зміни в одному форматі не призводять до конфліктів із кодом інших форматів.

На рівні окремого плагіна застосовано патерн «Стратегія», де інтерфейс Converter оголошує сімейство взаємозамінних алгоритмів конвертації, а кожен плагін надає власну їх реалізацію [28]. Це дозволяє конвеєру працювати з будь-яким форматом через спільний інтерфейс, не розрізняючи їх у коді; вибір конкретної стратегії відбувається під час виконання за псевдонімом формату, обраним користувачем. Поєднання мікроядерної архітектури (рівень системи) зі стратегією (рівень плагіна) забезпечує слабку зв'язаність компонентів та лінійну розширюваність.

Підтримка кожного формату файлів реалізована у вигляді окремого TypeScript-модуля (плагіна), розташованого у відповідній піддиректорії. Кожен плагін реалізує інтерфейс Converter, що складається з кількох полів-метаданих та двох методів перетворення; його склад наведено в табл. 2.2.

Таблиця 2.2.

Склад інтерфейсу Converter

Поле / метод	Призначення
fileExt: string	Унікальний короткий ідентифікатор формату (наприклад, "ust", "svr")
version: string	Версія плагіну
info: scoreInfo	Метадані формату (назва, автор, сайт, опис, розширення файлу)
inputOptions: OptionField[]	Опис параметрів для операції читання
outputOptions: OptionField[]	Опис параметрів для операції запису
import(data, options, warnings): Score	Метод читання: приймає байти файлу, повертає об'єкт Score
export(score, options): Uint8Array	Метод запису: приймає об'єкт Score, повертає байти файлу

Параметри читання та запису, на які посилаються поля inputOptions і outputOptions, описуються однотипними об'єктами OptionField. Кожен такий об'єкт декларативно задає одне налаштування конвертера – його ключ, тип,

значення за замовчуванням та текст для відображення у формі. Повний перелік полів об'єкта `OptionField` наведено в табл. 2.3.

Таблиця 2.3.

Поля об'єкта `OptionField`

Поле	Призначення
<code>name: string</code>	Ключ параметра у словнику <code>OptionsDict</code>
<code>type: OptionFieldType</code>	Тип ("bool", "int", "float", "string", "enum")
<code>default: OptionValue</code>	Значення за замовчуванням
<code>title: string</code>	Заголовок для відображення у формі
<code>description: string</code>	Опис

Завдяки цьому опису компонент `OptionsForm` може динамічно генерувати форму налаштувань для будь-якого плагіну, не маючи жорстко закодованих знань про конкретні формати. Тобто додавання нового параметра до плагіну автоматично призводить до появи відповідного елемента керування в інтерфейсі без жодних змін у коді самої форми.

Центральною точкою реєстрації є файл реєстру, що містить усі зареєстровані плагіни: `UST`, `USTx`, `CCS`, `SVP`, `VPR` та `VSQX`. Реєстр надає допоміжну функцію `getConverters`, яка повертає всі конвертери. Реєстрація є статичною та відбувається під час компіляції. Це є навмисним архітектурним рішенням для браузерного середовища: динамічне завантаження модулів у браузері ускладнює `tree-shaking` (видалення невикористаного коду збирачем пакетів), збільшує час ініціалізації та потребує обробки асинхронних помилок завантаження. Статичний реєстр дозволяє збирачу (`Next.js` + `Webpack/Turbopack`) включити у пакет лише необхідний код і гарантує, що при завантаженні сторінки всі конвертери вже готові до роботи.

Кожен плагін організований у власній піддиректорії та зазвичай складається з кількох файлів. Точкою входу є `index.ts`, що оголошує та експортує

об'єкт Converter, тоді як визначення масивів inputOptions і outputOptions винесено в options.ts. Внутрішню структуру формату описують TypeScript-інтерфейси у файлі model.ts, а власне логіку перетворення розподілено між parser.ts, що відповідає за розбір формату у внутрішнє представлення, і generator.ts, що виконує зворотне перетворення цього представлення у байти файлу. Такий поділ за відповідальністю робить кожен плагін самодостатнім і полегшує його тестування та супровід окремо від решти системи.

Наприклад, плагін UST зчитує текстовий файл у кодуванні, вказаному в параметрах (за замовчуванням UTF-8), парсить його у внутрішній об'єкт UstScore, потім перетворює цей об'єкт у канонічну модель. При записі послідовність зворотна: generateScore перетворює Score у UstScore, після чого exportUst() серіалізує його у текстовий рядок, що кодується у відповідне кодування. Порядок обробки наведено на рис. 2.3.

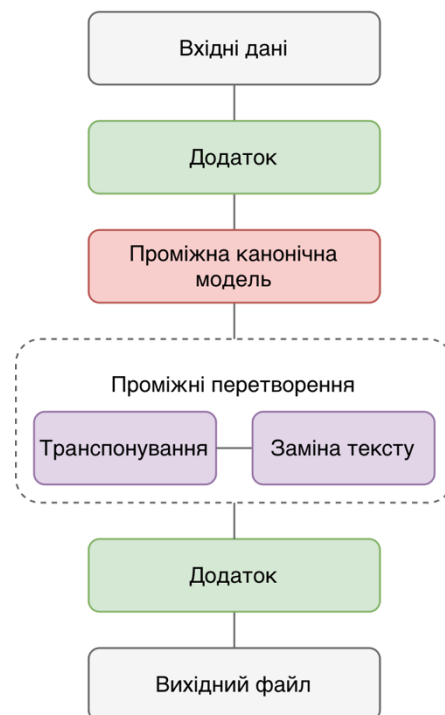


Рис. 2.3 Структура конвеєру обробки партитури та взаємодія компонентів

Ця структура означає, що підтримка нових форматів масштабується лінійно: кожен новий плагін додає рівно один конвертер до реєстру без будь-яких змін в архітектурі системи. Оскільки всі плагіни оперують єдиним типом

Score, новий плагін автоматично отримує сумісність з усіма вже наявними форматами.

2.5. Реалізація веб-інтерфейсу

Веб-інтерфейс реалізований за допомогою React-компонентів, організованих відповідно до системи маршрутизації App Router у Next.js. Кореневий файл layout.tsx визначає загальну оболонку HTML-документа: підключає шрифти й метадані сторінки, формує структуру `<html>` та `<body>`, а також містить невеликий inline-скрипт, що виконується ще до першого рендеру і встановлює клас `.dark` на кореневий елемент `<html>`, якщо відповідне налаштування збережено у `localStorage`. Винесення цієї логіки саме в синхронний inline-скрипт є свідомим рішенням: воно запобігає короткочасному «спалаху» невірної (світлої) теми, який інакше виник би між завантаженням розмітки та виконанням основного коду застосунку.

Інтерфейс побудовано за принципом композиції компонентів – складні елементи утворюються поєднанням простіших, кожен з яких має чітко окреслену відповідальність. Ієрархію компонентів наведено на рис. 2.4.

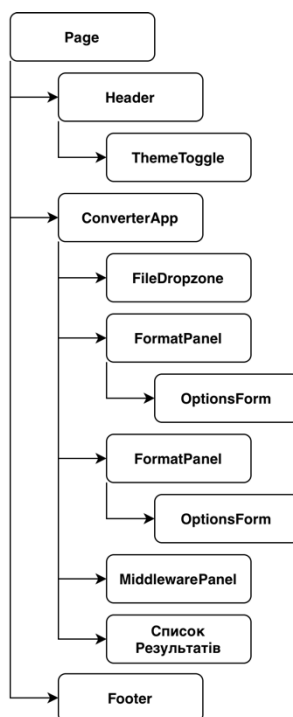


Рис. 2.4. Компонентна структура (ієрархія) веб-інтерфейсу

Структуру інтерфейсу утворює кілька компонентів. Шапку застосунку формує Header, що містить перемикач теми ThemeToggle. Головним контейнером, який утримує весь стан і координує дочірні компоненти, виступає ConverterApp. Усередині нього розміщені зона завантаження файлів FileDropzone, два екземпляри панелі FormatPanel (для вхідного та вихідного форматів), кожен з яких містить форму параметрів OptionsForm, а також панель проміжних перетворень MiddlewarePanel; список результатів конвертації при цьому рендериться безпосередньо у ConverterApp. Сторінка page.tsx є точкою входу для кореневого маршруту / і komponує три елементи: шапку <Header>, основний вміст <ConverterApp> та <Footer>. Це серверний компонент, що не потребує стану браузера – уся інтерактивна логіка зосереджена у <ConverterApp>, що дозволяє віддавати статичну оболонку сторінки максимально швидко.

ConverterApp є головним інтерактивним компонентом і позначений директивою "use client", оскільки інтенсивно використовує хуки React та браузерні API. Усі дані стану застосунку зосереджені саме в ньому, що робить його єдиним джерелом правди (single source of truth) для дочірніх компонентів: останні отримують потрібні дані через властивості (props) та повідомляють про дії через колбеки, але самі стану не зберігають. Такий однонаправлений потік даних спрощує відстеження причин кожної зміни інтерфейсу й унеможливорює узгодженість стану між компонентами. Зокрема, псевдоніми поточно обраних конвертерів для читання та запису зберігаються у inputAlias і outputAlias, а відповідні значення їхніх параметрів – у inputValues та outputValues (обидва типу OptionsDict). Перелік файлів у черзі на конвертацію міститься у масиві files: QueuedFile[], де кожен елемент має унікальний ідентифікатор (UUID) та об'єкт File; результати конвертації зберігаються у карті results, індексованій за id файлу, а прапор busy позначає, що конвертація триває, і блокує повторний запуск.

Для оптимізації продуктивності списки вхідних і вихідних конвертерів обчислюються через useMemo, щоб уникнути зайвих звернень до реєстру при кожному рендері, а обробники подій (addFiles, removeFile, selectInput,

selectOutput, convertAll) обгорнуті у useCallback, що запобігає непотрібному перестворенню функцій та зайвому повторному рендерингу дочірніх компонентів, яким ці функції передаються. У застосунку такого масштабу подібна мемоізація відчутно зменшує обсяг зайвої роботи браузера під час інтенсивної взаємодії користувача з інтерфейсом.

Окремо слід розглянути сценарій додавання файлів. Компонент FileDropzone підтримує два способи введення: перетягування файлів із файлового провідника безпосередньо у зону та вибір через стандартний системний діалог за натисканням. Обидва шляхи зводяться до єдиного обробника addFiles, який обгортає кожен доданий файл у структуру QueuedFile з власним UUID і додає його до черги; завдяки цьому подальша логіка не залежить від способу, у який файл потрапив до застосунку. Динамічні форми параметрів реалізує компонент OptionsForm, що будує елементи керування на основі описів OptionField обраного плагіну, через що додавання нового параметра до плагіну автоматично відображається в інтерфейсі без змін у коді самої форми.

При натисканні кнопки «Convert» викликається асинхронна функція convertAll. Вона послідовно ітерує по черзі файлів і для кожного спочатку зчитує його вміст у Uint8Array через File.arrayBuffer(), після чого викликає синхронну функцію convertFile(request). У разі успішної конвертації результуючий об'єкт Uint8Array та ім'я файлу зберігаються у стані, а якщо під час обробки виникає помилка – натомість зберігається її текст для відображення користувачу безпосередньо під відповідним файлом. Після кожного файлу стан results оновлюється, що викликає перерендеринг списку та відображення актуального статусу. Оскільки JavaScript у браузері є однопотокним, а функція convertFile – синхронною, між обробкою файлів відбувається неявне поступання контролю, завдяки чому React встигає оновити DOM між ітераціями, і користувач бачить, як статуси файлів змінюються по черзі, а не всі одночасно після завершення всього пакета.

Завантаження результату відбувається через програмне створення тимчасового елемента <a> з href = URL.createObjectURL(blob). Клік на цьому

елементі ініціює завантаження файлу браузером, після чого URL звільняється через `URL.revokeObjectURL`, аби уникнути витоку пам'яті. Цей підхід є стандартним способом ініціювати завантаження файлу, згенерованого в браузері, без звернення до сервера, що повністю узгоджується з клієнтською природою застосунку. Компонент `ThemeToggle`, своєю чергою, дозволяє перемикаати темний і світлий режими: поточна тема зберігається у `localStorage` під ключем `'theme'`, а при перемиканні клас `dark` додається або видаляється з `document.documentElement`, що миттєво змінює всі кольори інтерфейсу, адже Tailwind CSS 4 використовує стратегію `.dark`-класу, де кожен утилітарний клас може мати аналог із префіксом `dark`:

2.6. Реалізація проміжних перетворень

Проміжні перетворення (`middlewares`) – це компоненти системи, що маніпулюють канонічною моделлю `Score` після завершення фази читання (парсингу) та перед початком фази запису (серіалізації). Їх виділення в окремий етап конвеєра є важливим архітектурним рішенням: на відміну від плагінів форматів, кожне проміжне перетворення не залежить від конкретного формату й оперує виключно над уніфікованим представленням даних. Це дозволяє застосовувати одне й те саме перетворення незалежно від того, з якого формату читались та у який формат записуються дані, а отже, повністю усуває дублювання логіки обробки між плагінами. Якби, наприклад, транспонування реалізовувалося окремо в кожному плагіні, його довелося б підтримувати в шести місцях; натомість єдине перетворення над канонічною моделлю автоматично працює для всіх форматів.

Виконання ланцюга перетворень реалізує функція `applyMiddlewares`. Вона приймає об'єкт `Score` та масив конфігурацій перетворень, ітерує по ньому й послідовно застосовує лише увімкнені перетворення, причому кожне наступне отримує результат попереднього – таким чином утворюється конвеєр, у якому порядок елементів має значення. Порядок виконання визначається користувачем через інтерфейс і не є фіксованим. Така гнучкість важлива на практиці:

					ІІЗ.КР.Б. – 121 – 26 – ІІЗ	
						44

наприклад, якщо спочатку виконати конвертацію вимови (японська кана → ромаджі), а потім замінити лірику, правила заміни можуть спиратися на вже транслітерований текст – і навпаки, заміна лірики перед конвертацією вимови дасть інший результат. Кожне перетворення описується об'єктом-конфігурацією з унікальним ідентифікатором і набором параметрів, специфічних для кожного типу; за замовчуванням усі три перетворення вимкнені та можуть бути увімкнені й упорядковані користувачем незалежно. Наразі реалізовано три проміжні перетворення – транспонування нот, заміну лірики та конвертацію вимови, – які розглянуто нижче.

Транспонування нот зсуває висоту усіх нот вокальних доріжок на задану кількість напівтонів (key) і діє одночасно на двох рівнях. На рівні нот MIDI-номер кожної ноти змінюється на задану кількість напівтонів, що відповідає стандартному транспонуванню в музичній теорії. На рівні кривої висоти тону враховується, що крива зберігає відхилення у сентах відносно висоти відповідної ноти: оскільки опорна висота при транспонуванні змінюється, кожна точка кривої також зсувається на ту саму величину, перераховану із напівтонів у сенти (1 напівтон = 100 сентів). При цьому особливі сигнальні точки, що позначають відсутність питч-даних, навмисно залишаються незмінними, аби транспонування не «домалювало» криву там, де її не було. Якщо задане значення транспонування дорівнює нулю, перетворення повертає проєкт без жодних змін, уникаючи зайвих обчислень. Саме двоохрівнева дія робить транспонування коректним: зсув лише MIDI-номерів без відповідного зсуву кривої призвів би до розузгодження між нотами та їхньою інтонацією.

Заміна лірики застосовує до тексту лірики кожної ноти ланцюг правил заміни. Кожне правило (LyricsReplaceRule) задає шаблон для пошуку (pattern), рядок заміни (replacement), режим зіставлення (mode) та чутливість до регістру (caseSensitive, за замовчуванням вимкнена). Підтримуються чотири режими зіставлення: повний збіг рядка ("full"), збіг цілого слова ("alphabetic"), збіг підрядка ("non_alphabetic") та регулярний вираз ("regex"), у якому шаблон передається безпосередньо до RegExp. Функція compileRule(rule) збирає об'єкт

RegExp з відповідними префіксом та суфіксом залежно від режиму, прапором g (глобальна заміна всіх входжень) та необов'язковим прапором i (ігнорування регістру). Для режимів, що не є регулярними виразами, спеціальні символи екрануються функцією `escapeRegex` перед компіляцією – це запобігає ненавмисній інтерпретації символів як метасимволів регулярного виразу, через що, наприклад, крапка чи дужка у звичайному тексті лірики трактуються буквально. Ланцюг правил застосовується послідовно: кожне наступне правило отримує результат попереднього, утворюючи конвеєр послідовних заміन. Якщо правило містить некоректний регулярний вираз, функція перехоплює виняток і залишає текст незміненим – це запобігає аварійному завершенню конвертації через помилку у введених користувачем даних і відповідає вимозі стійкості системи до некоректного введення.

Конвертація вимови призначена для роботи з партитурами, що містять текст на японській або китайській (мандаринській) мові, і перетворює лірику кожної ноти відповідно до обраного режиму (`PronunciationMode`). Воно підтримує перетворення китайських ієрогліфів у пінїн ("`hanzi2pinyin`", тональна латинізація мандаринської мови) та японської кани у ромаджі ("`kana2romaji`"), а також приведення тексту до хірагани ("`to_hiragana`") чи катакани ("`to_katakana`"); режим "`none`" повертає лірику без змін [26] [27]. Для японськи x перетворень використовується бібліотека `wanakana` – JavaScript-бібліотека для роботи з японськими системами письма [20]. Для перетворення кани у ромаджі застосовується нестандартне мапінгове правило: っ (маленька «цу», що позначає гемінацію приголосного) $\rightarrow$ "cl", що відповідає фонемному позначенню у системах вокального синтезу, де подвоєння приголосного записується саме так. Особливим випадком є символ тривалого звуку: символи "ー", "—" та "-" (знак довготи у японській музиці) замінюються на ASCII-дефіс "-" незалежно від обраного режиму. Цей символ має спеціальне значення у партитурах вокальних синтезаторів – він позначає продовження попереднього складу – і тому повинен

залишатись нейтральним після будь-якого перетворення, аби не зруйнувати зв'язність співу.

Для перетворення ієрогліфів у пінїн реалізована власна функція `hanziToPinyin` на основі вбудованого словника `pinyinDict.json`. Словник містить відповідності між окремими ієрогліфами та їхніми найбільш уживаними безтональними пінїн-записами. Функція обходить рядок лірики посимвольно: ієрогліфи замінюються відповідниками зі словника, не-ієрогліфічні символи (латиниця, цифри, розділові знаки) додаються до буфера без змін, а між двома сусідніми алфавітно-цифровими частинами автоматично вставляється пробіл, що відокремлює склади один від одного. Власна реалізація замість зовнішньої бібліотеки обрана свідомо: вона не додає важких залежностей до клієнтського пакета й покриває потреби системи – безтональну латинізацію поширених ієрогліфів, достатню для подальшого синтезу. Усі три перетворення оперують виключно над канонічною моделлю, а тому повністю незалежні від форматів вхідного й вихідного файлів і можуть довільно комбінуватися користувачем.

2.7. Аналіз реалізованих плагінів форматів

У поточній реалізації системи підтримуються шість форматів партитур вокальних синтезаторів. Кожен реалізований у вигляді окремого плагіну з повною підтримкою як читання, так і запису, відповідно до інтерфейсу `Converter`. Попри спільний інтерфейс, плагіни суттєво відрізняються за внутрішньою складністю, що зумовлено принциповими відмінностями форматів у способах зберігання нот, висоти та кривої висоти тону. Розглянемо особливості реалізації кожного з них.

Плагін формату `UST` працює з текстовим файлом із секційною структурою (блоки `[#SETTINGS]`, `[#0000]`, `[#TRACKEND]`). Особливістю цього формату є відсутність абсолютних позицій нот: замість них зберігається лише тривалість кожної ноти, а позиція обчислюється як сума тривалостей усіх попередніх нот. Паузи між нотами позначаються спеціальними нотами з лірикою `"R"`. Важливо зазначити, що ця особливість формату не «протікає» у канонічну модель: ноти-

паузи з лірикою "R" не переносяться в неї як окремі об'єкти. На етапі парсингу їх тривалість лише додається до часового лічильника, через що наступна реальна нота отримує відповідно зміщену початкову позицію. У такий спосіб пауза перетворюється на властивість зміщення наступної ноти, а не на самостійний об'єкт моделі. Завдяки цьому проміжні перетворення та всі інші плагіни оперують виключно реальними нотами й повністю ізольовані від цієї специфічної особливості формату UTAU. Плагін підтримує декілька кодувань тексту (UTF-8, Shift-JIS) через параметр `encoding`, що важливо для японських партитур UTAU, традиційно збережених у Shift-JIS, адже хибне кодування зробило б лірику нечитабельною. Крива висоти тону зберігається у форматі дельт (`PitchBend`) і вимагає окремого алгоритму інтерполяції при парсингу та зворотного перетворення при генерації.

Плагін формату USTx обробляє проекти OpenUTAU – сучасної відкритої альтернативи UTAU. Файл зберігається у форматі YAML 1.2, що робить його легко читабельним для людини та зручним для версійного контролю, а для його парсингу використовується бібліотека `yaml` [19] [23]. На відміну від формату UST, USTx підтримує кілька вокальних та інструментальних доріжок одночасно. Інструментальні доріжки (`wavePart`) містять шлях до аудіофайлу та зміщення відтворення і при читанні перетворюються на об'єкти `InstrumentalTrack` канонічної моделі, тоді як вокальні доріжки (`voicePart`) організовані навколо масиву нот та масиву кривих автоматизації (`curves`), де кожна крива ідентифікується скороченим іменем параметра (`abbr`) і зберігає окремі масиви позицій (`xs`) та значень (`ys`). Структура ноти у USTx є більш деталізованою, ніж у UST: кожна нота містить власну криву висоти тону (`pitch`) у вигляді масиву точок із координатами `x`, `y` та типом інтерполяції між ними (`shape`), а також повний набір параметрів вібрато (`vibrato`) з глибиною, частотою, атакою, затуханням і фазовим зсувом. Плагін реалізує генерацію базової питч-кривої з нот через клас `BasePitchGenerator`, після чого додає до неї відхилення, збережені у `pitch.data`, – це дозволяє коректно відтворювати та генерувати плавні переходи

між нотами. Файл USTx може містити BOM на початку – плагін видаляє його перед парсингом, аби уникнути помилок розбору.

Плагін формату CCS є дещо складнішим, оскільки CeVIO зберігає висоту нот специфічно, замість MIDI-номера в XML-документі окремо зберігаються PitchOctave (октава) та PitchStep (ступінь у межах октави від 0 до 11) [22]. Конвертація між цим представленням і MIDI-номером при читанні та зворотним розкладанням при записі виконується за формулою

$$\text{keyNumber} = (\text{pitchOctave} + 1) \times 12 + \text{pitchStep}$$

Крива висоти тону у форматі CCS (LogF0) зберігається у логарифмічній шкалі частоти (Гц) і потребує перетворення у відхилення в сентах відносно нотної висоти при читанні та зворотного перетворення при записі; саме на цьому етапі застосовується перерахунок тіків у реальний час, оскільки CCS оперує абсолютними часовими мітками.

Плагін формату SVP має найскладнішу структуру серед оброблюваних форматів: це JSON-документ із групами нот (mainGroup, groups), референціями груп з параметрами трансформації та параметрами автоматизації у форматі масивів позицій і значень [21]. Відповідний плагін реалізує алгоритм злиття кількох груп в одну доріжку при читанні та розбиття назад при записі. Особливою складністю є обробка кривої pitchDelta, що зберігається відносно нотних висот у сентах і може мати різні режими інтерполяції (лінійна, крива Безьє та ін.), тому плагін реалізує парсинг параметрів pitchMode1 та pitchMode2 з відповідними алгоритмами інтерполяції. Саме для цього формату задіяно симулятор базової висоти, що відтворює плавні переходи між нотами (портаменто) і дозволяє коректно конвертувати відносну криву в абсолютну без спотворень, притаманних аналогам.

Плагін формату VPR працює з JSON-документом, упакованим у ZIP-архів, тому при читанні спочатку виконується розпакування архіву, а вже потім розбір JSON. Формат містить детальні параметри автоматизації через масив об'єктів controllers, де кожен об'єкт містить ім'я контролера та масив подій events, а список підтримуваних контролерів та їх маппінг до кривих канонічної моделі

задано окремим словником відповідностей, що дозволяє узгоджено зіставляти параметри VOCALOID із полями моделі.

Плагін формату VSQX завершує перелік – це XML-формат, що використовується у вокальних синтезаторах VOCALOID 3 та VOCALOID 4. Плагін підтримує читання обох версій формату (VSQ3 та VSQ4) з автоматичним визначенням версії за простором імен кореневого XML-елемента, а запис – виключно у формат VSQ4 як новіший. Файл VSQX може містити BOM або завершальні нульові байти – плагін очищує ці артефакти перед передачею тексту до XML-парсера, що забезпечує сумісність із файлами, збереженими різними версіями редактора VOCALOID. Формат організований навколо об'єкта masterTrack, що містить позначки темпу (tempo) та тактового розміру (timeSig), і масиву доріжок vsTrack, кожна з яких містить один або кілька блоків vsPart (вокальних груп) із масивами нот та точок кривих автоматизації (cc). Висота нот зберігається як MIDI-номер, що спрощує мапінг до канонічної моделі порівняно з форматом CCS. Криві автоматизації представлені масивами контрольних точок із позицією у тіках та значенням, аналогічно до формату VPR, тому плагін обробляє контролери VOCALOID через той самий словник мапінгу, що й плагін VPR, забезпечуючи узгодженість між обома форматами VOCALOID-сімейства.

Підсумовуючи, попри спільний інтерфейс Converter, плагіни демонструють різну внутрішню складність: від простого секційного текстового UST до багатогрупового JSON-формату SVP із відносною кривою висоти тону. Те, що всі вони взаємодіють із рештою системи виключно через канонічну модель, дозволяє приховати цю різноманітність за єдиним контрактом і додавати нові формати, не зачіпаючи наявні плагіни.

Висновки до другого розділу

У другому розділі виконано проектування та детальний опис архітектури та ключових компонентів системи обробки партитур вокальних синтезаторів.

У підрозділі 2.1 систему спроектовано як односторінковий вебдодаток (SPA), що функціонує повністю у браузері користувача без серверної обробки

даних. Для реалізації обрано стек TypeScript, Next.js, React та Tailwind CSS, що забезпечує статичну типізацію, декларативну реактивну модель оновлення інтерфейсу, підтримку темної теми та зрілу інфраструктуру збирання пакета. Принциповою характеристикою є повне виконання логіки парсингу, конвертації та серіалізації у рушії браузера, що усуває серверні залежності, спрощує розгортання та гарантує конфіденційність даних користувача.

У підрозділі 2.2 визначено функціональні вимоги: підтримка форматів UST, USTx, CCS, SVP, VPR та VSQX; збереження нот, темпу, тактового розміру та кривих висоти тону; пакетна обробка файлів; відображення статусу та попереджень; налаштування параметрів конвертерів; автоматичне визначення формату за розширенням. Нефункціональні вимоги охоплюють клієнтську обробку без завантаження даних, розширюваність через інтерфейс плагінів, швидкодію, точність конвертації та підтримку темної/світлої теми.

У підрозділі 2.3 детально описано канонічну проміжну модель, побудовану за принципом hub-and-spoke. Розкрито ієрархію класів Score → SingingTrack/InstrumentalTrack → Note та Params → ParamCurve → Point. Показано, що уніфіковане представлення кривої висоти тону з урахуванням переходів між нотами усуває недолік наявних аналогів.

У підрозділі 2.4 описано архітектуру плагінів: інтерфейс Converter з методами import та export, обґрунтовано вибір плагінної (мікроядерної) архітектури та патерну «Стратегія», розглянуто статичний реєстр конвертерів та структуру OptionField для декларативного опису параметрів.

У підрозділі 2.5 описано реалізацію веб-інтерфейсу: компонентну структуру (Header, ConverterApp, FileDropzone, FormatPanel, OptionsForm), управління станом через React-хуки, drag-and-drop завантаження файлів, динамічні форми параметрів, список результатів зі статусом та механізм генерації файлів через Blob API.

У підрозділі 2.6 детально описано три реалізовані проміжні перетворення: транспонування нот із синхронним зсувом кривої висоти тону; заміну лірики на основі ланцюга правил із чотирма режимами зіставлення та підтримкою

регулярних виразів; конвертацію вимови для японської (кана ↔ ромаджі/хірагана/катакана через бібліотеку wana kana) та китайської (ієрогліфи → піньін через вбудований словник) лірики. Порядок виконання перетворень визначається користувачем.

У підрозділі 2.7 проаналізовано реалізацію шести плагінів форматів: UST, USTx, CCS, SVP, VPR та VSQX.

РОЗДІЛ 3. ПРИКЛАД РОБОТИ ПРОГРАМИ

3.1. Інтерфейс користувача та порядок роботи з програмою

Програма реалізована у вигляді односторінкового вебдодатку, тому всі функціональні елементи інтерфейсу розміщені на одній сторінці та доступні користувачу без переходів між окремими екранами чи вкладками. Така організація відповідає основному сценарію використання – обробка файлів партитур, яка від початку до кінця виконується в межах єдиного робочого простору. Завдяки клієнтській архітектурі (вся обробка відбувається у браузері) інтерфейс залишається реактивним, а файли користувача не залежать від з'єднання з мережею після завантаження сторінки.

У верхній частині сторінки розташовано закріплену шапку застосунку, яка залишається видимою при прокручуванні. У лівій частині шапки знаходиться логотип програми разом із назвою «Converter» та підписом «Singing voice projects», що ідентифікують застосунок. У правій частині шапки розміщено індикатор «Client-side», який наголошує, що обробка відбувається на стороні клієнта, та перемикач теми оформлення – кнопка з піктограмою сонця або місяця, що дозволяє користувачу переключатися між світлою та темною темами інтерфейсу. Обраний режим зберігається у локальному сховищі браузера та відновлюється при наступному відкритті сторінки, що позбавляє користувача необхідності налаштовувати тему повторно. Інтерфейс програми показано на рис. 3.1.

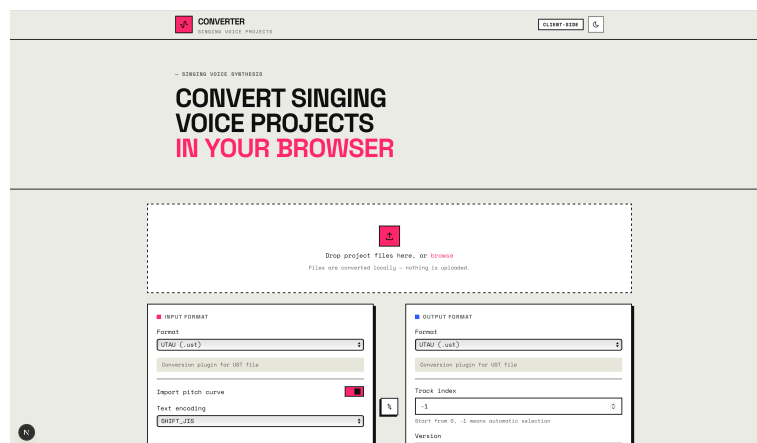


Рис. 3.1. Загальний вигляд інтерфейсу програми

Під шапкою розташовано заголовкову секцію з основним заголовком «Convert singing voice projects in your browser», що коротко повідомляє про призначення програми. Нижче починається основна робоча область, де зосереджені всі елементи керування процесом конвертації.

Першим елементом робочої області є зона завантаження файлів (рис. 3.2.), оформлена у вигляді прямокутника з пунктирною рамкою та піктограмою завантаження. Користувач може додати файли партитур двома способами: перетягнути їх із файлового провідника безпосередньо у цю зону або натиснути на неї та обрати файли через стандартний діалог вибору файлів операційної системи [12]. Підтримується додавання декількох файлів одночасно для пакетної обробки. У зоні завантаження також наявний напис «Files are converted locally – nothing is uploaded», який повідомляє користувачу, що файли не передаються на сервер, а обробляються локально. При наведенні файлу на зону її межа підсвічується, що візуально підтверджує готовність прийняти файл.

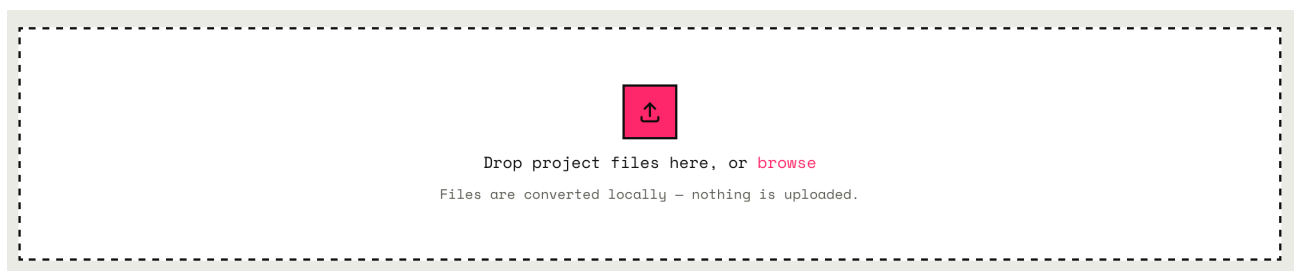


Рис. 3.2. Зона завантаження файлів

Нижче зони завантаження розміщено блок вибору форматів (рис. 3.3.), який візуально поділено на три частини: панель вхідного формату ліворуч, кнопка обміну форматами по центру та панель вихідного формату праворуч. Кожна панель формату містить випадаючий список, у якому користувач обирає формат партитури зі списку доступних плагінів. Кожен пункт списку відображає повну назву формату та його розширення. Під списком відображається короткий опис обраного формату. Кнопка обміну форматами, розташована між панелями, дозволяє одним натисканням поміняти місцями вхідний та вихідний формати, що зручно при зворотній конвертації.

Важливою особливістю є автоматичне визначення вхідного формату: при додаванні першого файлу програма аналізує його розширення та автоматично встановлює відповідний вхідний формат, тому в типовому випадку користувачу достатньо обрати лише цільовий (вихідний) формат.

Рис. 3.3. Панелі вибору вхідного та вихідного форматів

У нижній частині кожної панелі формату розташована форма параметрів, яка динамічно генерується залежно від обраного формату. Параметри відображаються у вигляді відповідних елементів керування: перемикачів для логічних значень, числових полів, текстових полів та випадаючих списків для переліку варіантів. Наприклад, для формату UST доступний параметр вибору кодування тексту, що дозволяє коректно обробляти японські партитури, збережені у кодуванні Shift-JIS. Якщо обраний формат не має параметрів, відображається відповідне повідомлення.

Під блоком вибору форматів розміщено панель проміжних перетворень (рис. 3.4.) із заголовком «Lyric & pitch transforms». Ця панель надає доступ до трьох перетворень, що застосовуються до партитури під час конвертації: зміни висоти тону (транспонування), заміни лірики та конвертації вимови. Перетворення подані у вигляді впорядкованого списку, де кожен елемент має перемикач увімкнення, кнопки зміни порядку (вгору/вниз) та порядковий номер. Поряд із заголовком панелі відображається лічильник активних перетворень. Підпис під заголовком пояснює, що перетворення застосовуються послідовно зверху донизу, а зміна їх порядку впливає на кінцевий результат.

При увімкненні кожного перетворення під ним розгортаються відповідні елементи налаштування. Для зміни висоти тону це числове поле, у якому

задається кількість напівтонів зсуву. Для конвертації вимови передбачено випадальний список вибору режиму перетворення – ієрогліфів у пінїн, кани в ромаджі, а також приведення тексту до хірагани чи катакани. Для заміни лірики відображається список правил заміни, де кожне правило складається з поля «From» (текст, що замінюється), поля «To» (текст підстановки), вибору режиму зіставлення (повний збіг, ціле слово, підрядок або регулярний вираз) та перемикача чутливості до регістру; правила можна додавати й видаляти, формуючи ланцюг послідовних замін.



Рис. 3.4. Панель проміжних перетворень

Під панеллю перетворень розташовано рядок дій. У його лівій частині відображається інформація про кількість доданих до черги файлів, а у правій – основна кнопка «Convert», що запускає процес конвертації. Якщо успішно конвертовано декілька файлів, поряд з'являється додаткова кнопка «Download all» для завантаження всіх результатів одночасно. Під час виконання конвертації кнопка змінює напис на «Converting...» та відображає індикатор завантаження.

Результати конвертації відображаються у вигляді списку під рядком дій (рис. 3.5.). Кожен елемент списку відповідає одному файлу та містить піктограму поточного стану (очікування, виконання, успіх або помилка), оригінальну назву файлу, назву результуючого файлу та кнопку «Download» для його

завантаження. Кнопка з піктограмою хрестика дозволяє видалити файл зі списку. У разі виникнення помилки під відповідним файлом відображається повідомлення з її описом, а у разі часткової втрати даних – блок попереджень, що інформує користувача про можливі неточності конвертації між форматами.

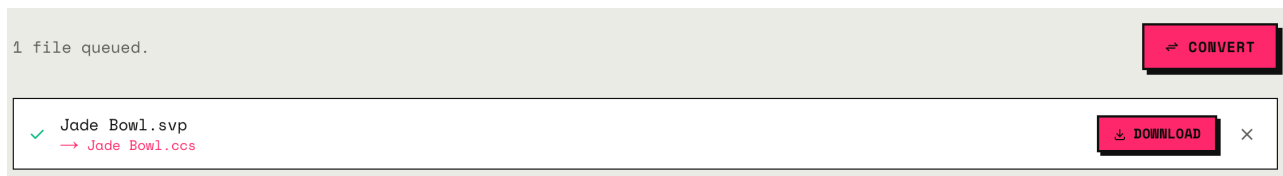


Рис. 3.5. Список результатів конвертації

Таким чином, типовий порядок роботи з програмою складається з таких кроків: користувач додає один або декілька файлів партитур у зону завантаження, за потреби коригує автоматично визначений вхідний формат та обирає цільовий вихідний формат, налаштовує параметри форматів і за бажанням вмикає проміжні перетворення, після чого натискає кнопку «Convert» та завантажує отримані файли. Весь процес виконується в межах однієї сторінки та не потребує встановлення додаткового програмного забезпечення.

3.2. Тестування та демонстрація роботи програми

Для підтвердження коректності роботи розробленої системи було проведено тестування, що складалося з двох частин: перевірки точності конвертації між усіма підтримуваними форматами та демонстрації роботи кожного з проміжних перетворень окремо.

Для першої частини тестування було підготовлено еталонну партитуру (рис. 3.6.), що містить характерний набір музичних даних: послідовність нот різної тривалості та висоти, текст лірики, призначений окремим нотам, позначки темпу та тактового розміру, а також криву висоти тону з вираженими переходами між нотами. Така партитура дозволяє перевірити збереження всіх ключових параметрів, що становлять найбільший інтерес при конвертації.

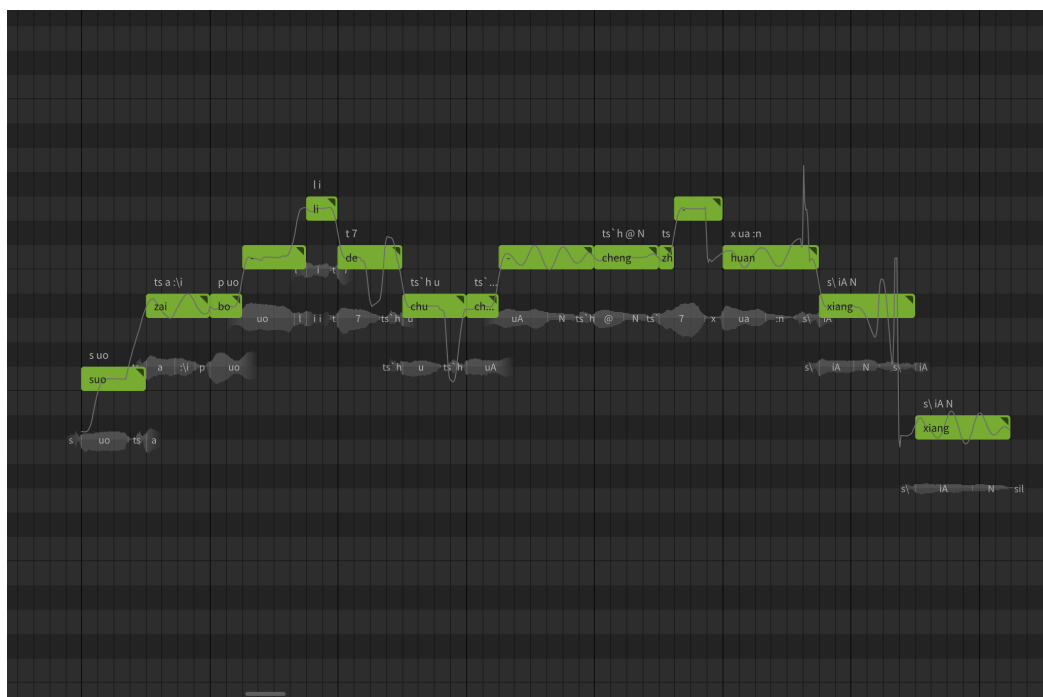


Рис. 3.6. Еталонна партитура у вихідному редакторі

Еталонну партитуру було послідовно конвертовано в усі підтримувані цільові формати – UST, USTx, CCS, VPR та VSQX. Для кожного результату конвертації виконувалося відкриття отриманого файлу у відповідному вокальному синтезаторі та візуальне порівняння з еталоном за такими критеріями:

- збереження кількості, позицій та тривалостей нот;
- збереження MIDI-висоти кожної ноти;
- відповідність тексту лірики, призначеного нотам;
- збереження позначок темпу та тактового розміру;
- збереження форми кривої висоти тону.

Результат конвертації еталонної партитури у формат UST показано на рис. 3.7. Файл відкрито в редакторі OpenUTAU, де видно, що послідовність нот повністю збережена: кількість, позиції та тривалості нот відповідають еталону, а кожній ноті призначено правильний склад лірики. Попри те, що UST зберігає позиції нот відносно (як суму тривалостей попередніх нот), реальне розташування нот на часовій сітці редактора збігається з вихідним, що підтверджує коректність обчислення абсолютних позицій під час запису. Крива висоти тону відтворює плавні переходи між нотами, відповідні оригіналу.

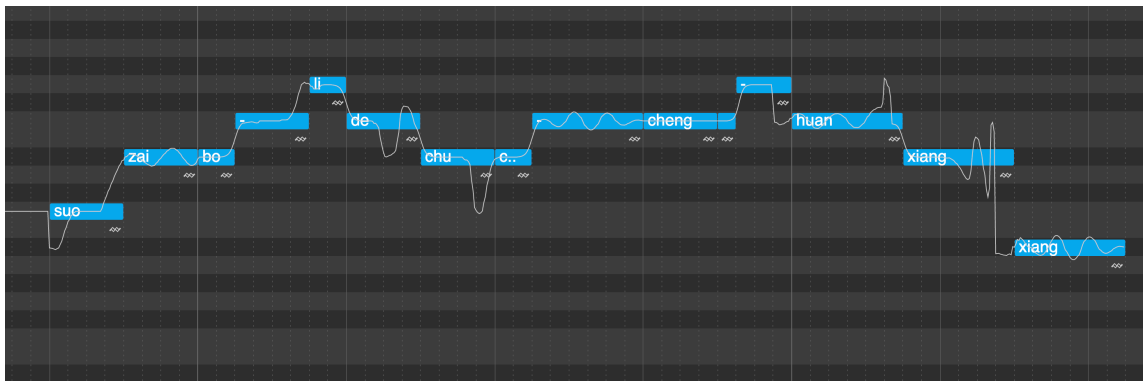


Рис. 3.7. Результат конвертації у формат UST (OpenUTAU)

Результат конвертації у формат USTx наведено на рис. 3.8. Оскільки USTx є рідним форматом OpenUTAU і має багатшу за UST структуру, у ньому збережено не лише ноти й лірику, а й детальнішу криву висоти тону з типами інтерполяції між точками. Порівняння з еталоном показує повну відповідність нот за позиціями, тривалостями та MIDI-висотою, а також коректне перенесення форми пітч-кривої, включно з переходами між сусідніми нотами.

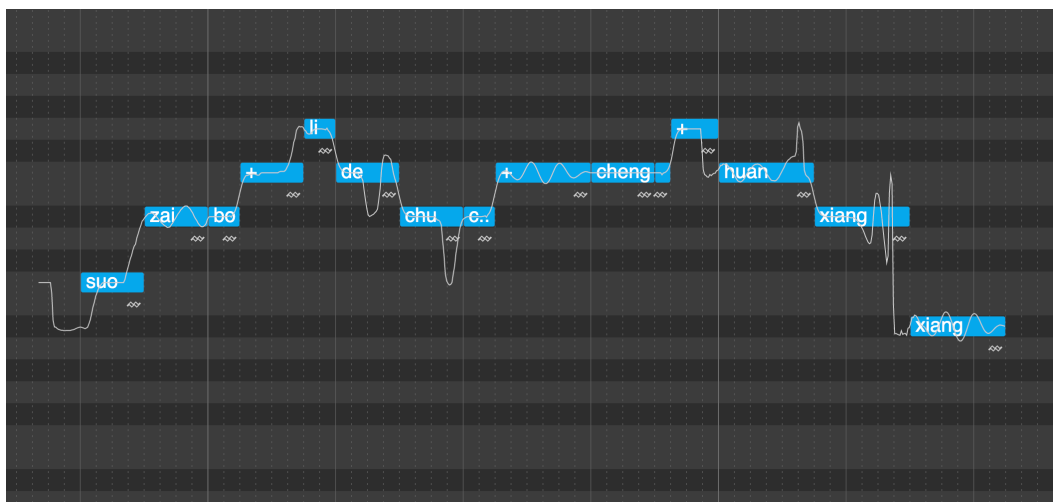


Рис. 3.8. Результат конвертації у формат USTx

На рис. 3.9 показано результат конвертації у формат CCS, відкритий у редакторі VoiSona. Цей формат становить особливий інтерес, оскільки зберігає висоту нот не MIDI-номером, а парою «октава–ступінь», а криву висоти тону – у логарифмічній шкалі частоти. Незважаючи на ці принципові відмінності представлення, ноти розташовані на тих самих позиціях і мають ту саму висоту, що й в еталоні, а крива висоти тону коректно відтворює інтонаційний контур, що підтверджує правильність перетворення між логарифмічним та канонічним (у сентах) представленнями.

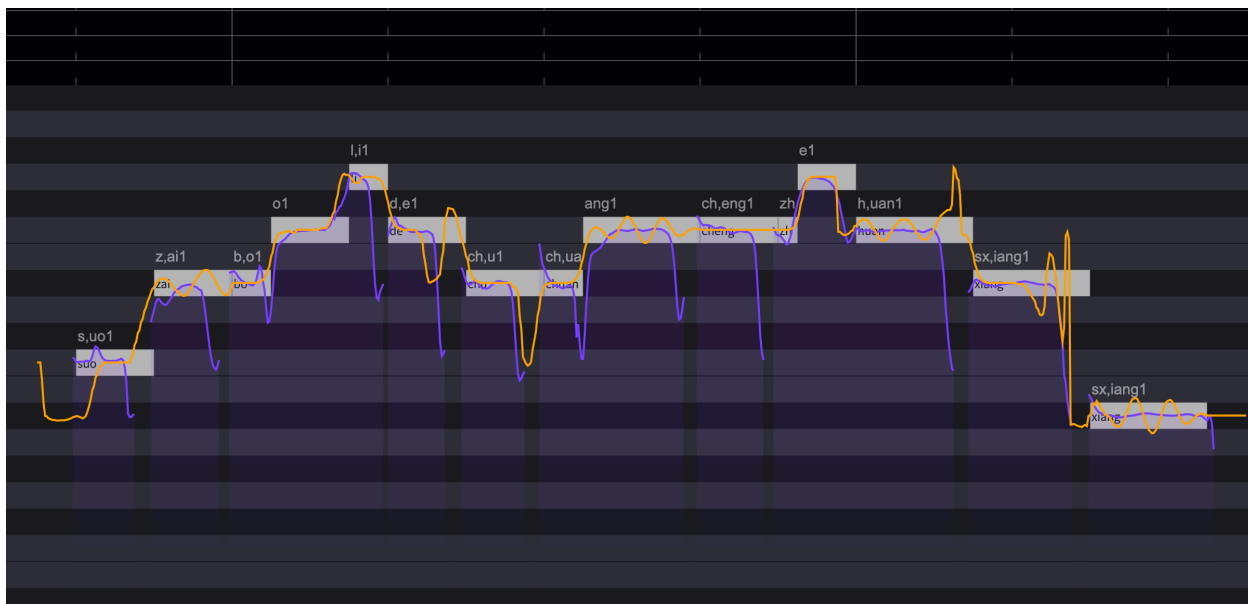


Рис. 3.9. Результат конвертації у формат CCS

Результат конвертації у формат VPR показано на рис. 3.10. Файл відкрито у VOCALOID, де видно, що ноти, їхні тривалості, лірика, позначки темпу й тактового розміру повністю відповідають еталонній партитурі. Оскільки VPR зберігає дані у вигляді JSON, упакованого в архів, цей результат додатково підтверджує коректність як формування внутрішньої структури, так і пакування файлу в очікуваний редактором контейнер.

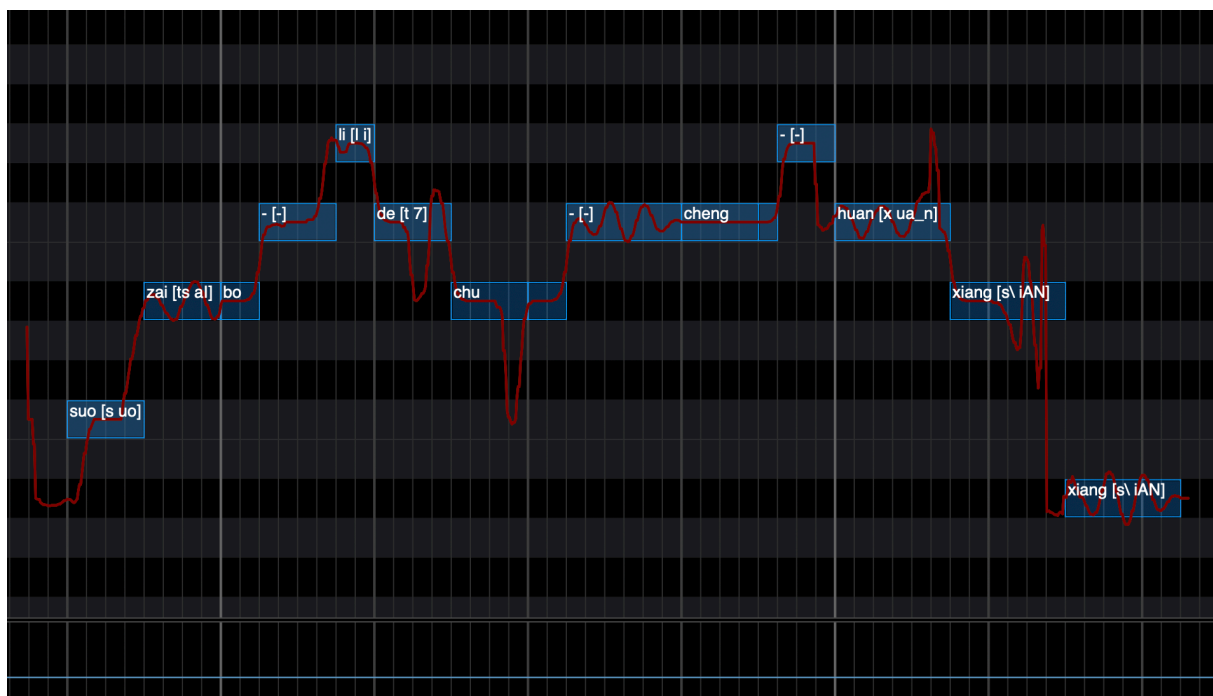


Рис. 3.10. Результат конвертації у формат VPR

На рис. 3.11 наведено результат конвертації у формат VSQX, відкритий у редакторі VOCALOID. Запис виконується у версію VSQ4, тож файл коректно розпізнається редактором, а всі ключові параметри – ноти, їхні позиції та висоти, лірика, темп і тактовий розмір – збігаються з еталоном. Крива висоти тону представлена контрольними точками у тіках і відтворює форму, задану у вихідній партитурі.



Рис. 3.11. Результат конвертації у формат VSQX (VOCALOID6)

За результатами порівняння встановлено, що в усіх випадках основні музичні параметри – ноти, їх позиції, тривалості та висоти, текст лірики, темп і тактовий розмір – переносяться без втрат. Крива висоти тону зберігається з урахуванням особливостей цільового формату: завдяки приведенню всіх кривих до єдиного представлення в проміжній моделі (відхилення у сентах відносно нотної висоти) переходи між нотами відтворюються коректно навіть при конвертації між форматами з принципово різними способами зберігання питч-даних – наприклад, між форматом SVP з відносною кривою та форматом CCS з абсолютною кривою.

Для другої частини тестування було підготовлено окрему партитуру, на якій послідовно й незалежно одне від одного продемонстровано роботу кожного

з трьох проміжних перетворень. Для кожного перетворення наводиться стан партитури до та після його застосування.

У панелі перетворень було увімкнено перетворення зміни висоти тону та задано зсув на визначену кількість напівтонів. Після конвертації всі ноти партитури виявилися зміщеними на задану величину зі збереженням ритмічної структури та лірики (рис. 3.12.). Разом із нотами було відповідно зміщено й криву висоти тону, що підтверджує коректність синхронного транспонування як самих нот, так і пов'язаної з ними автоматизації.

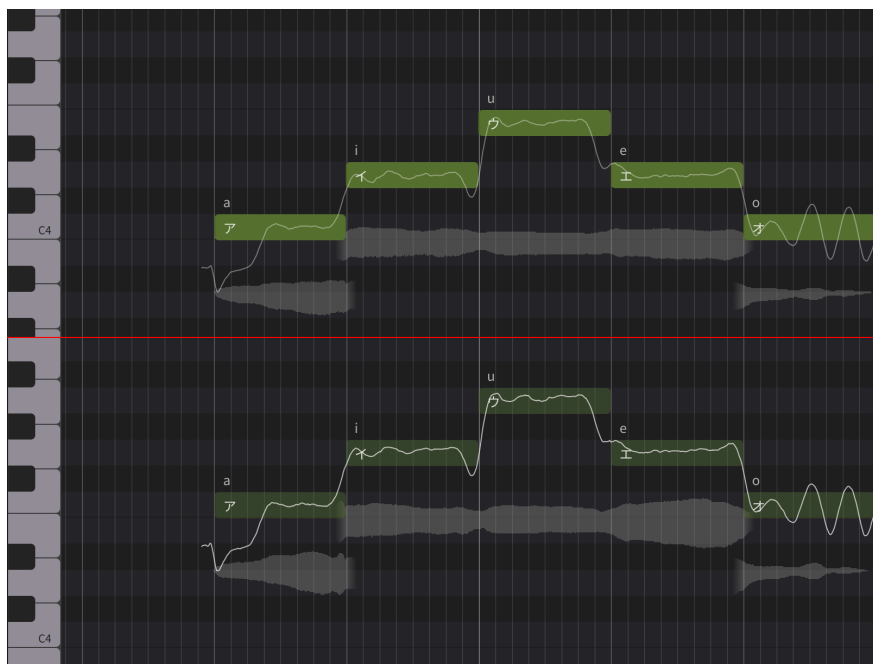


Рис. 3.12. Партитура до та після застосування транспонування

Для демонстрації заміни лірики було створено правило заміни, що задає текст для пошуку та текст для підстановки. Після конвертації відповідні склади лірики в нотах партитури було замінено згідно зі створеним правилом, тоді як решта музичних параметрів – ноти, їх висоти та тривалості – залишилися незмінними (рис. 3.13.). Заміна виконується послідовно для всіх входжень шуканого тексту, тому навіть повторювані склади опрацьовуються коректно й одноманітно по всій партитурі. При цьому структура нот залишається недоторканою: змінюється лише текстова мітка, прив'язана до ноти, а її часова та звуковисотна позиція зберігаються. Це підтверджує, що перетворення впливає виключно на текстову складову партитури.

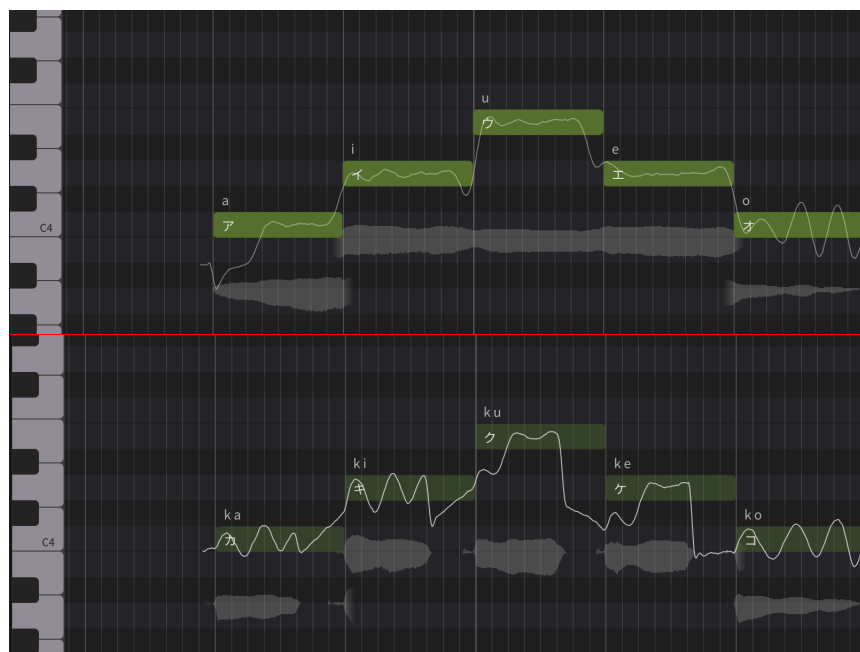


Рис. 3.13. Партитура до та після заміни лірики

Для демонстрації конвертації вимови було підготовлено партитуру з лірикою, записаною знаками японської абетки. У панелі перетворень було обрано відповідний режим конвертації, після чого текст лірики було автоматично транскрибовано у латинізовану форму – ромаджі для японської кани (рис. 3.14.). Така можливість є важливою при перенесенні партитур у синтезатори, що не підтримують введення тексту ієрогліфами, але приймають фонемні записи латиницею, і навпаки.



Рис. 3.14. Партитура до та після конвертації вимови

За результатами проведеного тестування підтверджено, що розроблена система коректно виконує конвертацію між усіма підтримуваними форматами зі збереженням ключових музичних параметрів, а також правильно застосовує проміжні перетворення партитури. Це свідчить про досягнення поставленої мети роботи – розширення сумісності цифрових партитур вокальних синтезаторів шляхом їх обробки та конвертації.

Висновки до третього розділу

У третьому розділі продемонстровано практичну роботу розробленої системи обробки партитур вокальних синтезаторів.

У підрозділі 3.1 описано інтерфейс користувача та порядок роботи з програмою. Показано, що завдяки односторінковій організації всі функціональні елементи – зона завантаження файлів, панелі вибору вхідного й вихідного форматів, форми параметрів, панель проміжних перетворень та список результатів – розміщені в межах єдиного робочого простору, а типовий сценарій конвертації виконується у кілька послідовних кроків без переходів між екранами.

У підрозділі 3.2 проведено тестування системи, що складалося з двох частин. У першій частині еталонну партитуру було конвертовано в усі підтримувані формати та підтверджено збереження ключових музичних параметрів – нот, темпу, тактового розміру, лірики та кривої висоти тону, зокрема при переході між форматами з принципово різними способами зберігання питч-даних. У другій частині продемонстровано коректну роботу кожного з трьох проміжних перетворень – транспонування, заміни лірики та конвертації вимови – окремо. Результати тестування підтверджують працездатність системи та досягнення поставленої мети роботи.

ВИСНОВКИ

У процесі виконання випускної кваліфікаційної роботи бакалавра здійснено аналіз предметної області та розробку системи обробки партитур вокальних синтезаторів, спрямованої на розширення сумісності цифрових партитур шляхом їх приведення до уніфікованого представлення. Поставлену мету та визначені завдання було досягнуто, а отримані результати мають як теоретичну, так і практичну цінність.

На першому етапі проведено порівняльний аналіз структур файлів партитур різних вокальних синтезаторів – UST, USTx, CCS, SVP, VPR та VSQX. Дослідження дозволило виявити принципові відмінності у способах зберігання музичних даних: відносне та абсолютне позиціонування нот, представлення висоти у вигляді MIDI-номера або пари «октава–ступінь», лінійні, логарифмічні та сплайнові криві висоти тону, а також різні текстові, XML, JSON та YAML контейнери. Виявлені особливості сформували вимоги до уніфікованого представлення даних.

На основі проведеного аналізу спроектовано канонічну проміжну модель партитури, побудовану за мережевим принципом hub-and-spoke. Модель зберігає ключову інформацію про доріжки, ноти, лірику, фонему, криві автоматизації, темп і тактовий розмір, нормалізуючи часові позиції у тіках незалежно від темпу. Застосування єдиного проміжного представлення дозволяє звести задачу взаємної конвертації N форматів до реалізації N незалежних відображень замість квадратичної кількості попарних перетворювачів, що забезпечує лінійну розширюваність системи.

Розроблено програмне забезпечення у вигляді клієнтського односторінкового вебдодатку на основі стека TypeScript, Next.js, React та Tailwind CSS. Принциповою відмінністю реалізації є повне виконання логіки парсингу, конвертації та серіалізації безпосередньо у середовищі браузера, без передачі даних на сервер. Такий підхід усуває серверні залежності, спрощує розгортання, гарантує конфіденційність даних користувача та забезпечує незалежність від мережевого з'єднання. Статична типізація мови TypeScript

усуває цілий клас помилок невідповідності типів між компонентами системи, що є істотним для коректної обробки складно структурованих музичних даних.

Архітектуру системи побудовано за модульним принципом із чітким розподілом відповідальностей між шарами представлення, стану, бізнес-логіки та плагінів. Підтримку кожного формату реалізовано у вигляді незалежного плагіну з уніфікованим інтерфейсом конвертації, а проміжні перетворення – транспонування нот, заміну лірики та конвертацію вимови – винесено в окремий конвеєр, що оперує над канонічною моделлю незалежно від форматів. Така організація дозволяє додавати нові формати та перетворення без модифікації наявних компонентів.

Працездатність та коректність системи підтверджено експериментально шляхом конвертації еталонної партитури в усі підтримувані формати зі збереженням музичних параметрів та демонстрації роботи кожного проміжного перетворення. Зокрема, показано коректне відтворення кривої висоти тону при переході між форматами з різними способами її зберігання, що усуває недолік, притаманний наявним аналогам.

Отримані результати мають практичне значення для музикантів, продюсерів та розробників програмного забезпечення у сфері вокального синтезу, забезпечуючи розширення сумісності цифрових партитур та підвищення ефективності роботи з ними. Розроблена система є розширюваною і має потенціал для подальшого розвитку – додавання підтримки нових форматів партитур та нових проміжних перетворень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bonada J., Serra X. Synthesis of the Singing Voice by Performance Sampling and Spectral Models. IEEE Signal Processing Magazine. 2007. Vol. 24, No. 2. P. 67–79. DOI: 10.1109/MSP.2007.323266.
2. Hono Y., Hashimoto K., Oura K., Nankaku Y., Tokuda K. Sinsy: A Deep Neural Network-Based Singing Voice Synthesis System. IEEE/ACM Transactions on Audio, Speech, and Language Processing. 2021. Vol. 29. P. 2803–2815. DOI: 10.1109/TASLP.2021.3104165.
3. Cho Y., Yang F., Chang J. A Survey on Recent Deep Learning-driven Singing Voice Synthesis Systems. arXiv preprint arXiv:2110.02511. 2021. URL: <https://arxiv.org/abs/2110.02511> (дата звернення: 14.06.2026).
4. VOCALOID – Official Website. URL: <https://www.vocaloid.com/en> (дата звернення: 22.06.2026).
5. Synthesizer V Studio – Dreamtonics. URL: <https://dreamtonics.com/synthesizerv/> (дата звернення: 09.06.2026).
6. CeVIO Creative Studio. URL: <https://cevio.jp/> (дата звернення: 17.06.2026).
7. UTAU – Vocal Synthesis Software. URL: <http://utau2008.web.fc2.com/> (дата звернення: 11.06.2026).
8. OpenUtau Documentation. URL: <https://github.com/stakira/OpenUtau/wiki> (дата звернення: 20.06.2026).
9. Synthesizer V Studio Scripting Manual. URL: <https://resource.dreamtonics.com/scripting/> (дата звернення: 08.06.2026).
10. UtaFormatix – Project Format Converter for Singing Voice Synthesizers. URL: <https://sdercolin.github.io/utaformatix3/> (дата звернення: 19.06.2026).
11. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 15.06.2026).
12. HTML Drag and Drop API. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API (дата звернення: 12.06.2026).

- 13.Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 21.06.2026).
- 14.React Documentation. URL: <https://react.dev/learn> (дата звернення: 10.06.2026).
- 15.Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 18.06.2026).
- 16.Node.js Documentation. URL: <https://nodejs.org/en/docs> (дата звернення: 07.06.2026).
- 17.File API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/File_API (дата звернення: 23.06.2026).
- 18.Blob. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Blob> (дата звернення: 13.06.2026).
- 19.yaml – JavaScript Parser and Stringifier Documentation. URL: <https://eemeli.org/yaml/> (дата звернення: 16.06.2026).
- 20.WanaKana – Japanese Romaji/Kana Conversion Library. URL: <https://wanakana.com/> (дата звернення: 06.06.2026).
- 21.Standard ECMA-404. The JSON Data Interchange Syntax. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-404/> (дата звернення: 22.06.2026).
- 22.Extensible Markup Language (XML) 1.0. W3C Recommendation. URL: <https://www.w3.org/TR/xml/> (дата звернення: 14.06.2026).
- 23.YAML Ain't Markup Language (YAML) Version 1.2 Specification. URL: <https://yaml.org/spec/1.2.2/> (дата звернення: 09.06.2026).
- 24.MIDI 1.0 Detailed Specification. The MIDI Association. URL: <https://midi.org/specs> (дата звернення: 18.06.2026).
- 25.MusicXML 4.0. W3C Community Group Report. URL: <https://www.w3.org/2021/06/musicxml40/> (дата звернення: 20.06.2026).
- 26.ISO 3602:1989. Documentation – Romanization of Japanese (kana script). URL: <https://www.iso.org/standard/8978.html> (дата звернення: 11.06.2026).

- 27.ISO 7098:2015. Information and documentation – Romanization of Chinese.
URL: <https://www.iso.org/standard/61420.html> (дата звернення: 17.06.2026).
- 28.Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1994. 395 p.
- 29.Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol, CA: O'Reilly Media, 2020. 432 p.
- 30.Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Sebastopol, CA: O'Reilly Media, 2019. 324 p.

ДОДАТКИ

					ПЗ.КР.Б. – 121 – 26 – ПЗ	

Порівняння аналогів та розробленого застосунку

Критерій	UtaFormatix	LibreSVIP	Synthesizer V (вбудована)	Розроблений застосунок
Спосіб доступу	Веб (браузер)	Десктоп	Усередині редактора	Веб (браузер)
Не потребує встановлення	Так	Ні	Потрібен Synthesizer V Srtudio	Так
Кросплатформність	Так	Так	Лише де працює Synthesizer V	Так
Кількість підтримуваних	Багато	Багато	Обмежено	6
Коректне збереження кривої	Частково	Частково	Не повністю	Так
Проміжні перетворення	Так	Так	Ні	Так
Пакетна обробка файлів	Так	Так	Ні	Так
Обробка без передавання даних	Так	Так	Так	Так

Словник основних термінів

Канонічна модель – уніфіковане проміжне представлення партитури, незалежне від конкретного формату файлу

Крива висоти тону – графічне представлення зміни висоти тону в часі

Парсинг – процес аналізу та розбору вхідних даних для їх подальшої обробки

Піньін – система латинізації китайської мови

Плагін – незалежний програмний модуль, що додає підтримку окремого формату партитур

Ромаджі – система латинізації японської мови

Серіалізація – перетворення об'єктів моделі даних у послідовність байтів для запису у файл

Тік – мінімальна одиниця вимірювання музичного часу; одна чверть ноти дорівнює 480 тікам

Транспонування – зміна висоти всіх нот на задану кількість напівтонів

Фонема – найменша одиниця звуку в мові