Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка застосунку «Музична школа»: розробка інструментів для формування основних елементів музики»
за освітньою програмою: «12 Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1912»

_____ /Валентин ДЄЄВ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /ст. викл. Ірина ШАПОВАЛ/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /Світлана ВОЛКОВА/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Дніпро – 2023 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of the «Music school» app: basic music elements instruments development»
according to educational curriculum « Software engineering »
in the Speciality: «121 Software engineering»

Done by the student of the group PZ: / /

Scientific Supervisor: /Iryna SHAPOVAL/

Normative controller: / /

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

На кваліфікаційну роботу
бакалавра студенту
1. Тема роботи: «Розробка застосунку «Музична школа»: розробка інструментів для формування основних елементів музики»
Керівник роботи: Шаповал Ірина Вікторівна, старший викладач
затверджені наказом № 77 ст від 07.12.2002.
2. Строк подання студентом роботи: 19.06.2023
3. Вихідні дані до роботи:
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати): вступ, збір вимог до програмного забезпечення, зовнішнє і внутрішнє проектування, тестування та налагодження, висновки, література.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентація, відео-демонстрація роботи програми.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	03.04.2023 – 05.04.2023	5%
2	Огляд літератури та аналіз аналогів	06.04.2023 – 09.04.2023	10%
3	Розробка структур вхідних і вихідних даних	10.04.2023 – 24.04.2023	20%
4	Визначення вимог до програми. Вибір та обґрунтування мови програмування	25.04.2023 – 02.05.2023	25%
5	Узгодження та затвердження ТЗ	03.05.2023 – 10.05.2023	30%
6	Розробка та програмування логіки програми	11.05.2023 – 18.05.2023	50%
7	Розробка і реалізація інтерфейсу користувача	19.05.2023 – 26.05.2023	70%
8	Відлагодження програми	27.05.2023 – 02.06.2023	80%
9	Розробка, узгодження та затвердження програмної документації	03.06.2023 – 18.06.2023	95%
10	Подання кваліфікаційної роботи до кафедри	19.06.2023 – 26.06.2023	100%
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	27.06.2023	100%

Студент _____ _____
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ ст. викл. Ірина ШАПОВАЛ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

- вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 1 сторінки;
- збір вимог до програмного забезпечення – у цьому розділі описуються аналоги програми та література по даній предметній області, а також проводиться опит зацікавлених сторін для формування найбільш повних вимог до програмного забезпечення. Складається з 10 сторінок;
- зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка фізичного проекту, приводиться опис об'єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи, вибір мови програмування. Складається з 31 сторінки;
- тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорної» та «білої» скриньки. Також аналіз помилок їх вплив на систему та вирішення проблеми. Складається з 18 сторінок;
- висновки. Складається з 2 сторінок;
- список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;
- додатки – містить робочий проект додаткового без .txt файлу та зображень для форм.

Кількість таблиць: 11 штук.

Кількість рисунків: 8 штук.

Ключові слова: нота, тональність, лад, півтон, тон, ступінь, знак альтерації, нотний стан, октава, мажор, мінор, кастомізація, тоніка.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ (програмне забезпечення) – програмний продукт, який застосовується на комп’ютерних системах таких як персональні комп’ютери, ноутбуки, мобільні пристрої, планшети, телефони а також деколи і на різній побутовій техніці на кшталт пральних машин, мікрохвильовок, холодильників та інше.

Кастомізація – підлаштовування, налаштування чогось під себе. Те ж саме що й персоналізація.

Нота – ступінь або звук, що має характеристики: октава, знак альтерації, протяжність та інші.

Тональність – це висотне розташування ладу. Усього існує 30 тональностей.

Лад – це сукупність звуків, які на основі спорідненості між ними об’єднані в систему, що має тоніку.

Тоніка – базова нота тональності або ладу, тобто та нота, від якої йде побудова тональності чи ладу.

Півтон – одиниця виміру музичних інтервалів, є найменшою відстанню між звуками за висотою. Зазвичай, візуально на фортепіано, у ролі півтону виступає одна клавіша.

Тон – одиниця виміру музичних інтервалів, містить в собі два півтони. Зазвичай, візуально на фортепіано, у ролі тону виступає дві клавіші підряд.

Ступінь – це звук музичної системи (звукоряду). Усього основних ступенів сім – «до», «ре», «мі», «фа», «соль», «ля», «сі».

Знак альтерації – знак, що відображає підвищення або зниження ступеню на півтона або тон. Таких знаків декілька: дієз (♯), дубль-дієз, бемоль (♭), дубль-бемоль, а також бекар (♮) – знак відміни дій інших знаків.

Нотний стан – це система з п’яти паралельних горизонтальних ліній, на якій розміщують ноти.

Октава – інтервал від «до» до «сі» включаючи обидві ноти, відображає звукоряд певної висоти. Таких октав може бути на різних інструментах різну кількість, і усі октави мають власні назви та власну висоту звучання.

Мажор – музичний лад, аккорди якого будуються на великій терції. Зазвичай цей лад має позитивне, радісне звучання.

Мінор – музичний лад, аккорди якого будуються на малій терції. Зазвичай цей лад має похмуре, сумне звучання.

ВСТУП

Музика, як і у минулому, сьогодні, так і надалі залишатиметься невід'ємною частиною нашого життя. Технології просунулася вперед і тим самим змінилася сама музика – з'явились безліч нових інструментів та різноманітних жанрів, покращилась якість звучання, спростився процес створення нових композицій, пісень, аранжувань. Але будь-який інструмент у руках людини без гарної практики та базових знань – просто не розкритий інструмент, на якому неможливо відтворити щось дійсно приголомшливе. У всіх сферах є умовна "база знань", де потрібно досягти максимального рівня навичок, щоб отримати фундамент, завдяки якому людина стане майстром справи. У музиці важлива кожна нота, зіграна на інструменті, адже кожна нота задає тон та настрій композиції. Але все ж таки, чому музика настільки важлива для нашого світу? А важлива вона тим, що завдяки їй часто люди піднімають собі настрій, задають атмосферу у своєму повсякденному житті, мають різні спогади, пов'язані з окремими піснями. В тяжкі для людини часи, музика надає натхнення рухатись вперед та направляє на різноманітні вчинки, надає мотивацію, підтримує дух людини аби не дати їй впасти духом та поринути у депресію. Словом, «Музика дозволяє рухатись вперед і бути закоханим у неї». А ще є такі люди, як меломани, що просто не уявляють своє життя без музики, адже меломани обожнюють абсолютно різні жанри.

Завдяки технологіям нашого часу можна знайти різноманітні програми, що допомагають музикантам у вивченні та виявленні особистих проблем у музиці. Такі програми допомагають музикантові у його становленні, як висококваліфікованого фахівця. Тому, аби допомогти усім музикантам, а саме: як початківцям, так і більш досвідченим, – було вирішено створити ПЗ «Elegia», яка сприятиме у відточенні музичних навичок та знань музичної теорії, методом використовування запропонованих інструментів ПЗ для досягнення почуття, що передає та сама фраза: «Музика допомагає рухатися вперед…».

Завданням цієї дипломної роботи є вдосконалення навичок знань музиканта. Робота передбачає за собою написану музичну програму, що має тренажерні тести та інструментарій для музикантів із простим інтерфейсом для легкого використання. Для користувача ПЗ, здебільшого, є два режими у кожному розділі – режим тренування та режим інструменту, завдяки яким він зможе відточити свої знання в музичній теорії на практиці. Показувати результат тестування користувачеві – важлива складова навчання. Окрім зміцнення знань, «Elegia» матиме ряд інструментів із роботою для піаніно. Користувачем може бути будь-яка зацікавлена у музиці людина – як музикант-початківець, так і професійний музикант зі стажем.

1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Перегляд аналогових конкурентів

З усіх переглянутих конкурентів можна виділити інтернет-ресурси «www.musictheory.net» і «www.musicca.com». Вони схожі за своє метою – зміцнити навички музикантів, але різні за відношенням до користувача. В обох аналогів великий функціонал тестів і тренажерів, але більшу частину функціоналу має «www.musictheory.net».

1.1.1. Огляд продукту «musictheory»

Musictheory – представляє з себе сайт теоретичних та практичних уроків для визначення нот та інших інструментів із кастомізацією режимів. Сама підготовка музиканта полягає в тому, що користувач бачить перед собою простий інтерфейс із нотами, де можна побачити саму ноту та варіанти відповіді до неї. Суть режиму “Визначення нот” полягає в тому, щоб вибрати правильну назву щодо зображеної ноти, де також можна відстежувати кількість визначених нот, а при неправильному виборі кількість спроб вгадування збільшується, але відсоток вірних нот падає з кожним неправильним кліком на варіант. (приклад інтерфейсу наведено на рис. 1.1.).

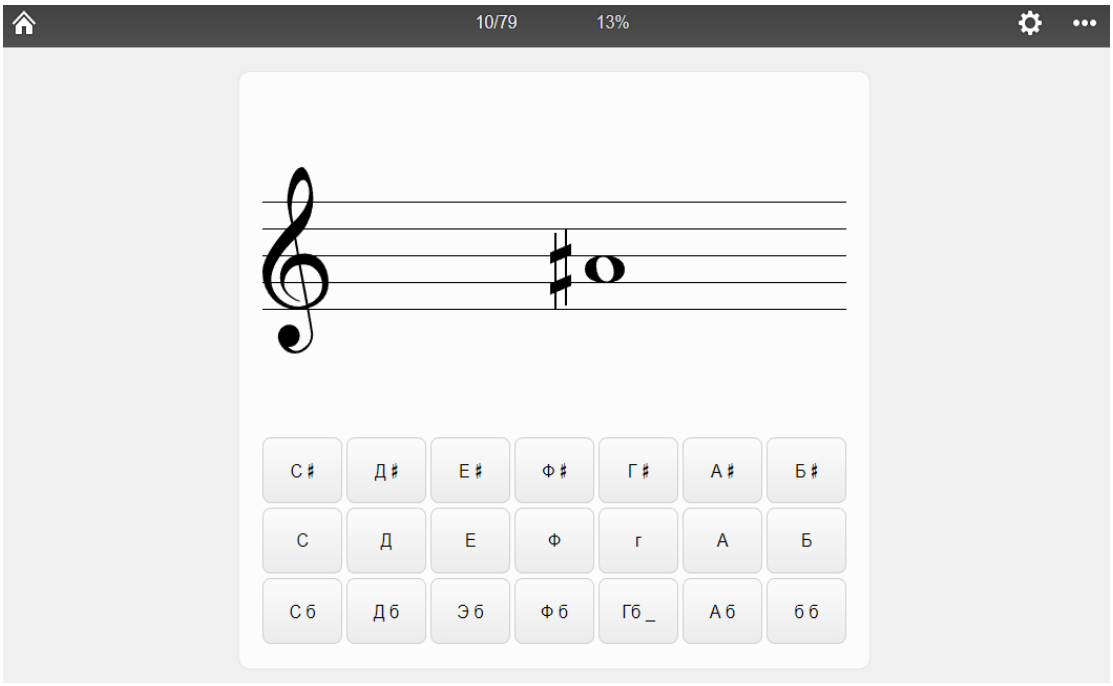

Рисунок 1.1. Приклад інтерфейсу «Musictheory»

Переваги:

- Хороший функціонал;
- Кастомізація режимів на смак користувача;
- Анімовані уроки;
- Музичні інструменти (калькулятор);

Недоліки:

- Немає регулювання гучності твору ноти;
- Немає маркування клавіш;
- Доволі однотонний дизайн;

Веб-сайт має великий функціонал, а самому користувачеві доволі цікаво користуватися можливістю кастомізації режимів, де можна налаштувати деякі деталі нот та інші налаштування. Головний мінус сайту є простий однотонний дизайн без яскравих кольорів. Кнопка результатів не має яскравого виділяючого кольору, тому користувачеві складно що-небудь помітити. Є уроки які анімаційно розповідають про музичну теорію, а також є вправи з обмеженим вибором інструментів.

1.1.2. Огляд продукту «musicca»

Musicca - музична платформа, що надає уроки, вправи, і також інтерактивні інструменти, які можуть допомогти музикантові більше дізнатися про музику і покращити його навички.

Загалом музична платформа Musicca позиціонує себе як веб-сервіс інструментарію та тренажерів, що дозволяють відточити навички як з читання нот, тональностей на клавіатурі фортепіано, так і використання квінтового кола. Цей веб-сервіс можна назвати найбільш насиченим за функціоналом, та найбільш візуально (по інтерфейсній частині) привабливим для користувача, адже має певного роду мінімалістний вигляд та дизайн, що у сучасні часи є найбільш популярним рішенням як для веб-сервісних інтерфейсів, так і для додатків для мобільних та стаціонарних пристроїв (сам приклад інтерфейсу наведено на рис. 1.2).

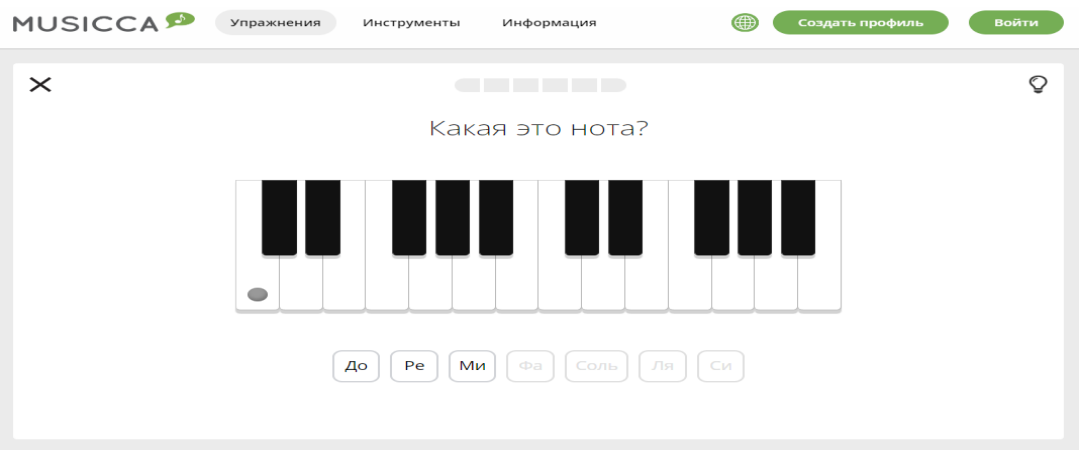

Рисунок 1.2. Приклад інтерфейсу «Musicca»

Переваги:

- Непоганий функціонал;
- Уроки, що використовують частину інтерфейсу прямо на сторінці;
- Музичні інструменти (калькулятор);
- Простий та зручний дизайн;
- Локалізація багатьма мовами;

Недоліки:

- Регулювання гучності нот;
- Відсутня кастомізація режимів;

Сподобалося, що можна залишати маркерування на тих нотах яких захоче користувач, що є дуже зручним використанням. Візуально дизайн сходиться зі стилістикою піаніно і не сильно виділяється, що для сайту є хорошою ознакою, тому що під час роботи деякі кольори можуть виділятися.

1.2. Огляд літератури

1.2.1. Базова музична теорія.

Для того, щоб правильно читати та писати, потрібно знати слова, розуміти їх сенс і граматику, тобто певні закони мови. Те саме із музикою - щоб вміти правильно створювати музику, грати її або просто співати – потрібно розуміти теорію музики.

Музика складається із нот. Нота – це звук певної висоти. Багато з нас знає назви нот ще зі школи, такі як: «до», «ре», «мі», «фа», «соль», «ля», «сі».

Усі ці сім основних нот (або ж ступенів) утворюють октавну систему – система повторюємих нот, кожні сім з яких, починаючи від «до» та закінчуючи на «сі» утворюють одиницю цієї октавної системи – октаву. Кожна октава має власну навзу і, як уже було описано, повторючі ноти, які відрізняються одна від одної за висотою звучання.

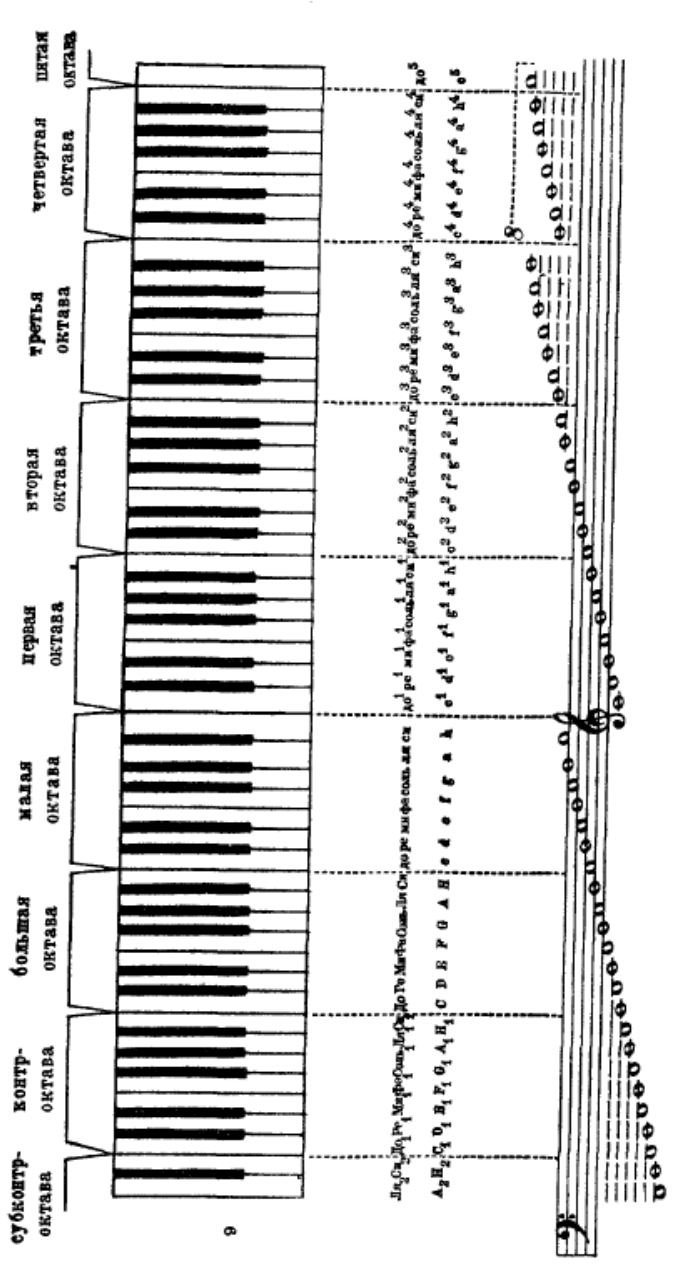

Рисунок 1.3. Загальна октавна система

Наприклад, нота «до» малої октави і нота «до» другой октави будуть відрізнятись за звучанням в тому сенсі, що перша буде звучати нижче, аніж друга. Усього октавна система налічує, зазвичай, 9 октав. Інші, теоретично,

можуть також існувати, проте зовсім не використовуватись у музиці, або ці ж інші октави практично неможливо почути людським вухом. Так, на рис. 1.3 можна побачити перелік октав у октавній системі: субконтроктава, контроктава, велика октава, мала октава, перша октава, друга октава, третя октава, четверта октава, п’ята октава.

1.2.2. Нотний стан

Проте усі ці складові музичної теорії безпосередньо відображаються на нотному стані – способі зображення музичної інформації, а саме послідовності нот, завдяки якій можна зіграти певну композицію. Якщо говорити точніше, нотний стан – це система з п’яти паралельних горизонтальних ліній, на яких розміщуються ноти. Розрахунок ліній ведеться знизу вверх.

Нотний стан на своєму початку має ключ – знак, який ставиться на одній з лінії нотного стану. Усього таких знаків існує декілька, але з них є два основні – скрипковий та басовий. Кожен з них ідентифікує ноту, на якій він знаходиться. Тобто для скрипкового – це нота «соль» першої на другій лінії нотного стану, для басового – це нота «фа» малої октави на четвертій лінії нотного стану. Від цього ключа завжди відраховуються усі інші ступені, що знаходяться на лініях та між ними.

Кожна нота, що знаходиться на нотному стані (рис. 1.4), розташовується на лінії або між лініями. Від її розташування залежить її висота. Тобто, чим вище на нотному стані намальована нота, тим вище за звучанням нота.

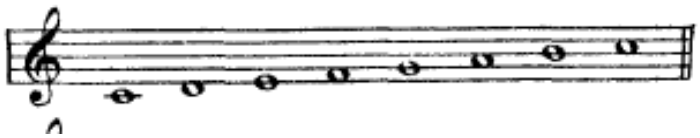

Рисунок 1.4. Скрипковий нотний стан та ноти на ньому – від «до» до «сі»

Також існують знаки альтерації. Знак альтерації – це знаки, що розташовуються зліва від ноти. Такі знаки вказують підвищення або пониження основної ступені, якій відповідає нота. Всього існує п’ять знаків альтерації: дієз (♯), дубль-дієз, бемоль (♭), дубль-бемоль, бекар (♮). Бемолі означають пониження ступені, дієзи – підвищення. Бекар означає відміну

попередньо застосованого знаку для ноти. Простими словами – означає звичайну ступінь, без підвищення чи пониження.

1.2.3. Тональність.

Насправді, сама тема тональностей є доволі складеною темою для викладення у даній роботі. Тому у ПЗ, створеному в рамках дипломної роботи, тональність використовується лише для її побудови та відображення на клавішах фортепіано.

Перед тим як зрозуміти, що являє собо тональність, потрібно знати що означає лад. Лад – це сукупність звуків, які на основі спорідненості між ними об’єднані в систему, що має тоніку. Тоніка – головний стійкий звук системи, простими словами – це перша нота у ладу. Взагалі в основному виділяють два лади – мажор та мінор. Мажорний лад має більш жваве, веселе та оптимістичне звучання, а мінорний – сумне, пригнічене, деколи епічне звучання.

Тому, тональність – це висотне розташування ладу. Різні тональності мають різні назви, що складаються з двух частин – тоніки та визначення ладу. Тобто, якщо взяти тональність «C♯ мажор», то «до дієз» – це тоніка, а «мажор» – це лад.

Усього теоретично існує 30 тональностей. Більша частина тональностей має паралельні тональності – парні тональності натурального мажора та мінора. Такі тональності мають однакові ключові знаки. Певна частина тональностей практично не використовується через те, що вони мають проблемні моменти для застосування.

1. До-бемоль мажор (паралельна – Ля-бемоль мінор);
2. Соль-бемоль мажор (паралельна – Мі-бемоль мінор);
3. Ре-бемоль мажор (паралельна – Сі-бемоль мінор);
4. Ля-бемоль мажор (паралельна – Фа мінор);
5. Мі-бемоль мажор (паралельна – До мінор);
6. Сі-бемоль мажор (паралельна – Соль мінор);
7. Фа мажор (паралельна – Ре мінор);
8. До мажор (паралельна – Ля мінор);

9. Соль мажор (паралельна – Мі мінор);
10. Ре мажор (паралельна – Сі мінор);
11. Ля мажор (паралельна – Фа-дієз мінор);
12. Мі мажор (паралельна – До-дієз мінор);
13. Сі мажор (паралельна – Соль-дієз мінор);
14. Фа-дієз мажор (паралельна – Ре-дієз мінор);
15. До-дієз мажор (паралельна – Ля-дієз мінор).

Ноти, що зіграні в межах певної тональності, завжди звучатимуть доречно та гармонійно. Тобто, можна сказати що одна, але не остання, з ролей тональності – це надання бачення виконавцю гармонічних зон під час написання або програвання музики.

Висновки до пункту 1

У музиці найбільш головну та невід'ємню роль грають ноти. Адже без нот неможливо написати жодний твір, зіграти жодну мелодію. Ноти присутні у будь-якій музиці.

Необхідність застосування тональностей зазвичай пояснюється тим, що сам по собі твір, що не має певної тональності буде звучати дуже негарно, так як буде відчуватися дисонанси під час програвання мелодії, що робить пісню не лише поганою за звучанням, а й неграмотно побудованою. Тому, можна сказати, що і тональність сама по собі є ключовою у музиці.

2. ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

2.1. Зовнішнє проектування

2.1.1. Функціональне призначення

Функціональним призначенням програми є відточення навичків музикантів-початківців і любителів. Серед таких навичок: читання нот з нотного стану, тренування музичного слуху, навички зі знання тональностей.

2.1.2. Експлуатаційне призначення

Експлуатаційне призначення програми:

- роль музичного інструментарію;
- підвищення рівня базових знань у музиці;
- самоперевірка своїх знань;
- розвиток сприйняття нот на слух;

2.1.3. Функціональні вимоги

Дана музична програма має виконувати такі види вимог як:

- відтворювати тестові випробування з метою розвитку навичок для обраної користувачем категорії:
  - ✓ Note Identifier – ідентифікатор нот, навички читання з нотного стану;
  - ✓ Tonality Builder – побудова тональностей. Відточує навички зі знання різних тональностей, а також є інструментом з їх побудови;
- будувати візуальну схему із точок на віртуальному фортепіано у категорії Tonality Builder;
- кнопки режимів вибору налаштувань;
- відображати результати кожного циклу тестів;
- трекінг помилок кількості вірних відповідей та кількості тестів.

2.1.4. Вхідні та вихідні дані

2.1.4.1. Вхідні дані

Вхідними даними є:

1) isJustBeginned – змінна, що позначає чи виконується генерація тестових випробувань для користувача вперше у циклі. Дана змінна є типу bool та застосовується в обох функціоналах – Note Identifier та Tonality Builder;
2) note – змінна, що позначає ноту-відповідь від користувача під час режиму тренування у Note Identifier. Ця нота звіряється із нотою, згенерованою цим функціоналом. Дана змінна є типу перечислення Degree, що позначає основні ступені (C, D, E, F, G, A, B).
3) index – змінна, що використовується у методі визначення існування перечислення Degree за наданим індексом. Дана змінна є типу uint8_t. Допустимі значення – від 0 до 255.
4) scaleRequestData – змінна, що застосовується у функціоналі Tonality Builder, та вміщує в собі загальні дані про тональність. Дана змінна є типу структури ScaleRequestData. Ця структура містить такі змінні: degree_ типу Degree – ступінь ноти, що є тонікою тональності; alterationSign_ типу AlterationSign – знак альтерації тоніки; key_ типу Key – лад тональності (мажор чи мінор).
5) generatedScale – змінна, що позначає згенеровану тональність. Опираючись на цю змінну, також на інтерфейсі будується точкова схема на фортепіано для візуалізації тональності. Вона використовується під час генерації тональності для уникнення генерації одинакових тональностей. Дана змінна є типу структури Scale. Ця структура містить такі змінні: tonic_ типу Note – тоніка тональності (зі ступенем degree_ типу Degree, знаком альтерації alterationSign_ типу AlterationSign, октавою octave_ типу Octave); key_ типу Key – лад тональності (мажор чи мінор); notes_ типу NoteVec – вектор (динамічний масив) нот, що знаходяться в даній тональності; enharmonicScale_ типу NoteVecOpt – вектор нот енгармонічної тональності, тобто такої, що по факту є такою самою тональністю і містить такі самі ноти, але просто називається по іншому. Opt позначає, що такої енгармонічної тональності може і не бути.

6) tonalityExists – змінна, що позначає існування згенерованої тональності. Вона допомагає уникнути генерації неіснуючих тональностей. Дана змінна є типу bool.
7) noteVec – змінна, що позначає вектор нот, за яким треба сформувати тональність. Використовується під час пошуку інформації про тональність за відомим переліком нот. Дана змінна є типу вектор нот Note.
8) volume – змінна, що відповідає за гучність нот, що програються через клас AudioEngine.

2.1.4.2. Вихідні дані

Вихідними даними є:

1) GenerateIdentifyNote – метод, що повертає згенеровану ноту для функціоналу Note Identify. Повертаємо значення є типу строка, що з боку інтерфейсу інтерпретується у вигляді розміщенного спрайту ноти на екрані.
2) IsCorrectAnswer – метод, що перевіряє надану користувачем відповідь на правильність. Цей метод використовується як у Note Identify, так і Tonality Builder функціоналі. Повертаємо значення є типу bool.
3) GetCorrectAnswers – метод, що повертає кількість вірних відповідей наданих користувачем. Цей метод використовується як у Note Identify, так і Tonality Builder функціоналі. Повертаємо значення є типу uint8_t (від 0 до 255).
4) GetNotesGenerated – метод, що повертає кількість згенерованих нот у Note Identify функціоналі. Повертаємо значення є типу uint8_t (від 0 до 255).
5) GenerateTonality – метод, що повертає згенеровану тональність для функціоналу TonalityBuilder. Повертаємо значення є типу Scale, який містить в собі згенеровану тональність та всі дані про неї.

6) BuildTonality – метод, що повертає побудовану тональність на основі даних, що надаються методу, про необхідну тональність. Повертаєме значення є типу Scale.
7) GetScalesGenerated – метод, що повертає кількість згенерованних тональностей у Tonality Builder функціоналі. Повертаєме значення є типу uint8_t (від 0 до 255).
8) PlayNote – метод, що програє звук певної ноти на певній гучності. Повертаємого значення не має, а лише відтворює звук на персональному комп'ютері.
9) StartAndPlayQueue – метод, що програє звуки декількох нот у наданій черзі, послідовно відтворючи звуки на персональному комп'ютері. Повертаємого значення не має.

2.1.5. Опис зовнішнього інформаційного середовища

Програма, для повноцінного функціонування, потребує:

– Операційна система: Windows 7 або новіша;
– Наявність стандартних бібліотек Visual C++;
– Наявність динаміків для відтворення звуку нот;
– Наявність бібліотек QT 6.4.0 або новіше.

Нижче наведена діаграма прецедентів що зображує специфікацію функціональних вимог (рис. 2.1).

На ній можна побачити, що користувач має можливість використовувати функціонали нотного ідентифікатора (Note Identifier, тільки режим тренування), будувальника тональностей (Tonality Builder, обидва режими тренування та інструменту), транспонувальника (Transposer, обидва режими тренування та інструменту).

У кожному функціоналі користувач може програвати звуки нот під час роботи, але у нотному ідентифікаторі за замовчення ноти програються після генерації одразу. Цю функцію можна буде вимикати натисканням кнопки виключення звуку на інтерфейсі під час роботи.

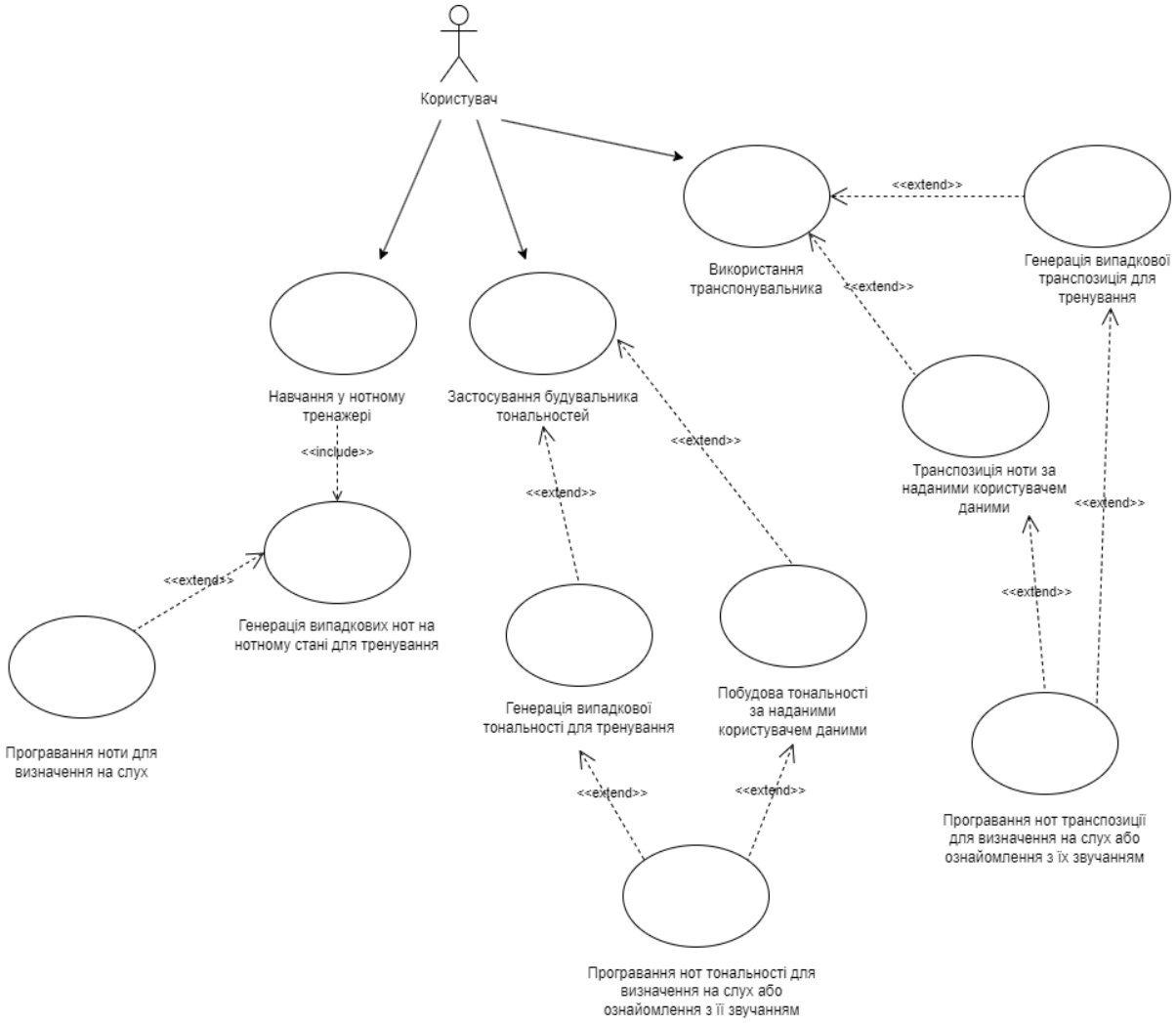

Рисунок 2.1. Діаграма прецедентів

Транспонування у даній дипломній роботі розглядатися не буде, так як цю частину виконав студент Лисенко Максим (див. дипломну роботу студента Лисенка Максима).

2.2. Внутрішнє проектування

2.2.1. Аналіз зовнішніх специфікацій систем

2.2.1.1. Моделювання словника системи

Серед усіх сценаріїв взаємодї користувача можна виділити такі сутності як: Note Identifier (ідентифікатор нот), Tonality Builder (будувальник тональностей), Audio Engine (аудіо процессор), Note (сутність ноти), Scale (сутність тональності), Random (рандомізатор), Note Identifier Controller (допоміжна сутність зі сторони інтерфейсу для ідентифікатора нот), Tonality

Builder Controller (допоміжна сутність зі сторони інтерфейсу для будувальника тональностей).

Опис обов’язків сутностей:

1) Note Identifier – реалізує функціонал системи вгадування нот, а саме логічну частину, обробку відповідей та генерації нот;
2) Tonality Builder – реалізує функціонал побудови тональностей, включаючи режим навчання та режим інструментарію. Виконує обробку відповідей користувача, обробку запитів на побудову тональності, генерацію тональностей для режиму навчання;
3) Audio Engine – відтворює звуки нот та нотних послідовностей (наприклад, ноти тональності), що дозволяє користувачу чути ноти під час генерації нот у Note Identifier, а також під час натискання на кнопки програвання у Tonality Builder;
4) Note – зберігає повну інформацію про ноту. Серед такої інформації:
    a. Ступінь (C, D, E, F, G, A, B);
    b. Знак альтерації (Дієз. Бемоль, бекар, дубль-дієз, дубль-бемоль або немає);
    c. Октава (суб-контроктава, контроктава, велика октава, мала октава, перша октава, друга октава, третя октава, четверта октава, п’ята октава, шоста октава);
5) Scale – зберігає повну інформацію про тональність. Серед такої інформації:
    a. Тоніка (ступінь, що відіграє роль першої ступені у тональності, від якої будується тональність);
    b. Знак альтерації тоніки;
    c. Лад;
    d. Перелік нот тональності;
    e. Перелік нот енгармонічної тональності – різною за назвою, але однаковою по факту звучання;

6) Random – керує рандомізацією чисельних значень, тобто генерацією випадкових чисел. Ці випадкові числа використовуються під час генерації нот та тональностей у режимі навчання.
7) Note Identifier Controller – оперує логікою Note Identifier зі сторони інтерфейсної частини користувача, обробляючи відповіді від Note Identifier і формуючи їх на екрані.
8) Tonality Builder Controller – оперує логікою Tonality Builder зі сторони інтерфейсної частини користувача, обробляючи відповіді від Tonality Builder і формуючи їх на екрані.

Детальні відповідні атрибути та операції, що стосуються кожного класу та є необхідними для виконання обов'язків наведено ниже, у табл. 2.1.

Таблиця 2.1. – Сутності, атрибути та методи

Сутність	Атрибути	Методи
1	2	3
Note Identifier	correctAnswers_ – лічильник вірних відповідей; generatedNote_ – поточна згенерована випадкова нота; notesGenerated_ – лічильник згенерованих нот.	GenerateIdentifyNote – генерує випадкову ноту для подальшої ідентифікації користувачем під час випробування; IsCorrectAnswer – перевіряє відповідь користувача на правильність; GetCorrectAnswers – повертає кількість вірних відповідей, даних користувачем; GetNotesGenerated – повертає загальну кількість генерацій нот;

Продовження таблиці 2.1

1	2	3
		NullifyData – аннулює кількість генерацій та кількість правильних відповідей перед початком кожного нового випробування; IsDegreeEnumIndex – визначає, чи існує за наданим індексом нотна ступінь.
Tonality Builder (у коді ScaleBuilder)	correctAnswers_ – лічильник вірних відповідей; generatedScale_ – поточна згенерована тональність для режиму тренування; scalesGenerated_ – лічильник згенерованих тональностей.	GenerateTonality – генерує випадкову тональність для режиму тренування; BuildTonality – утворює тональність, опираючись на вхідні дані по необхідній тональності; IsCorrectAnswer – перевіряє надану відповідь користувачем на правильність; GetCorrectAnswers – повертає кількість вірних відповідей; GetScalesGenerated – повертає кількість згенерованих тональностей;

Продовження таблиці 2.1

1	2	3
		GetScaleRequestData – повертає опис тональності за наданим переліком нот; GetScale – повертає тональність опираючись на вхідні дані, а саме опис тональності (тоніка, знак альтерації, лад); GetEnharmonicScale – повертає енгармонічну тональність, а саме перелік нот цієї тональності, пов’язаної вхідними даними – описом тональності, за яким шукається енгармонічна тональність. IsSuccessfulGeneration – перевіряє генерацію на успішність, звіряючи існування згенерованої тональності із записаними у базі функціоналу, тим самим запобігаючи помилкам генерації; NullifyData – аннулює кількість генерацій та кількість вірних відповідей

Продовження таблиці 2.1

1	2	3
		перед початком кожного нового випробування; ValidateScaleRequestData – перевіряє запит що містить опис тональності, на коректність. Тобто, виконує валідацію.
Audio Engine	mediaPlayer_ – клас, що відповідає за програвання аудіофайлу; audioOutput_ – клас, що відповідає за вивід звуку під час програвання аудіофайлу; notesQueue_ – утримує чергу з нот, звук яких потрібно програти послідовно; lastVolumeLevel_ – зберігає останній рівень гучності, який застосовувася останній раз (від 0.0 до 1.0); audioDuration_ – відповідає за тривалість програвання аудіофайлу (у секундах).	AudioEngine – конструктор, ініціалізує необхідні атрибути для роботи початковими значеннями; PlayNote – програє задану ноту на заданій гучності. StartAndPlayQueue – програє задану послідовну чергу з нот на заданій гучності; StopAudio – зупиняє програвання будь-якого аудіо у програмі; GetFilePathForNote – повертає шлях до аудіо файлу, що містить звук заданої ноти; PlayAudioFile – програє аудіо файл за заданим шляхом на заданій гучності; PlayNextAudio – програє

Продовження таблиці 2.1

1	2	3
		наступний аудіо файл із черги. Виконує роль тріггера.
Note	degree_ – ступінь ноти; alterationSign_ – знак альтерації; octave_ – октава.	GetNextDegree() – повертає наступну ступінь за висотою після поточної ступені даної ноти; GetPreviousDegree() – повертає попередню ступінь за висотою перед поточною ступінню даної ноти; GetNextOctave() – повертає наступну октаву за висотою після поточної октави даної ноти; GetPreviousOctave() – повертає попередню октаву за висотою перед поточною октавою даної ноти; IsSignGreater() – порівнює чи є перший вхідний знак альтерації більшим за другий вхідний знак альтерації; IsSignLesser() – порівнює чи є перший вхідний знак альтерації більшим за

Продовження таблиці 2.1

1	2	3
		другий вхідний знак альтерації.
Scale	tonic_ – позначає ноту-тоніку тональності; key_ – лад (мінор чи мажор); notes_ – перелік нот в даній тональності; enharmonicScale_ – перелік нот енгармонічної тональності (якщо є).	
Random		RandomizeInt32 – виконує генерацію випадкового цілого числа від першої вхідної межі до другої.
Note Identifier Controller	uiPtr_ – слугує як смарт-вказівник на головне вікно програми; audioEnginePtr_ – слугує як смарт-вказівник на аудіо процессор Audio Engine; isEndless_ – позначає, чи користувач обрав нескінченний режим тренування;	NotesIdentifierController – конструктор, слугує ініціалізатором контроллера функціоналу; BeginTheFlow – починає процес тренажеру ідентифікатору нот; ProceedAnswer – оброблює відповідь користувача; IsSoundOn – перевіряє, чи присутній звук у функціоналі та повертає стан;

Продовження таблиці 2.1

1	2	3
	isSoundOn_ – позначає, чи користувач увімкнув звук у даному функціоналі; maxAmount_ – зберігає значення кількості генерацій, обраної користувачем.	SetSoundState – змінює стан роботи звуку у функціоналі; SetNotePosOnSheet – розміщує згенеровану ноту на нотному стані на інтерфейсі користувача; SelectedIndexToNoteAmount – конвертує індекс комбо-боксу для вибору максимальної кількості нот.
Tonality Builder Controller	uiPtr_ – слугує як смарт вказівник на головне вікно програми; audioEnginePtr_ – слугує як смарт-вказівник на аудіо процессор Audio Engine; currentTonality_ – поточна згенерована тональність. isEndless_ – позначає чи користувач обрав нескінченний режим тренування; isToolMode_ – позначає, чи зараз активований режим інструменту;	TonalityBuilderController – конструктор, слугує ініціалізатором контроллера функціоналу; BeginLearnFlow – ініціалізує процес тренування у функціоналі; BeginToolFlow – ініціалізує процес інструментарію у функціоналі; ControlBuildNextButtonState – активує-деактивує кнопку побудови або кнопку продовження тренування; IsToolMode – визначає, чи зараз активовано режим інструментарію та повертає

Продовження таблиці 2.1

1	2	3
	maxAmount_ – зберігає значення кількості генерацій, обраної користувачем;	відповідне значення; ProceedAnswer – оброблює відповідь користувача у режимі тренування; BuildTonality – будує тональність за запитом користувача у режимі тренування; PlayBuiltTonality – програє побудовану тональність, програючи звуки нот, що входять у неї; ControlPlayButtonState – активує-деактивує кнопку програвання звуків нот тональності; SetTonalityDotsOnKeys – розміщує точки на віртуальному фортепіано, візуально зображуючи тональність для користувача; HideDotsOnKeys – приховує побудовані точки від користувача; HideShowLearningModeElements – ховає або показує

Закінчення таблиці 2.1

		інтерфейсні елементи навчання (наприклад лічильники вірних відповідей) SelectedIndexToTonalityAmount – конвертує індекс комбо-боксу кількості тональностей для визначення у режимі тренування; ValidateTonality – корректизує тональність, а саме регуляє перехід на іншу октаву нот, що виходять за неї, і таким чином розміщує їх на віртуальному фортепіано коректно.

2.2.1.2. Моделювання розподілу обов’язків у системі

Нижче наведено перелік різних множин класів, що співпрацюють одне з одним задля виконання певних процесів:

– Note Identifier Controller, Note Identifier – передача згенерованої ноти на інтерфейс, прийняття та обробка відповідей від користувача, робота функціоналу ідентифікатора нот;
– Note Identifier Controller, Audio Engine – програвання звуку згенерованих нот;
– Note Identifier, Note – формування ноти;
– Note Identifier, Random – рандомізація ноти для генерації;

- Tonality Builder, Audio Engine – програвання звуку нот побудованих та згенерованих тональностей;
- Tonality Builder, Scale – формування тональності;
- Tonality Builder, Random – рандомізація тональності для генерації;
- Audio Engine, Note – програвання звуку конкретної ноти;
- Scale, Note – утворення послідовності нот у тональності;
- Tonality Builder Controller, Tonality Builder – передача згенерованої або побудованої тональності на інтерфейс, прийняття та обробка відповідей чи запитів побудови тональності від користувача, робота функціоналу будувальника тональностей;
- Tonality Builder Controller, Audio Engine – програвання звуку нот тональностей;

Примітивні типи будуть визначені під час побудови діаграми класів. Сам результат моделювання зв’язків різного виду наведено нижче у табл. 2.

Таблиця 2.2. – Моделювання залежностей

Клас, котрий зв’язується	Клас із котрим зв’язується	Тип зв’язку
1	2	3
Note Identifier Controller	Note Identifier	Реалізація
	Audio Engine	Асоціація
Note Identifier	Note	Композиція
	Random	Асоціація
Tonality Builder	Scale	Композиція
	Random	Асоціація
Audio Engine	Note	Асоціація
Scale	Note	Композиція
Tonality Builder Controller	Tonality Builder	Реалізація
	Audio Engine	Асоціація
Note	–	–

Закінчення таблиці 2.2.

Random	–	–

2.2.1.3. Визначення призначень об’єктів за допомогою CRC карток

Призначення об’єктів перелічено із застосуванням CRC (Class-Responsibility-Collaboration) карток нижче у таблицях 2.3 – 2.10. Завдяки таким карткам можна одразу побачити які класи походять від яких, які класи мають певних нащадків, а також

Таблиця 2.3. – CRC-картка для класу «Note Identifier Controller»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Опрацьовує запити користувача під час роботи з Note Identifier, отримує результати логічних операцій від Note Identifier класу та інтерпретує їх на графічному інтерфейсі.	Note Identifier, Audio Engine

Таблиця 2.4. – CRC-картка для класу «Note Identifier»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Реалізує логічні операції щодо генерації нот, перевірку відповідей, що приходять від Note Identifier Controller.	Note, Random

Таблиця 2.5. – CRC-картка для класу «Tonality Builder»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні

Закінчення таблиці 2.5.

Обов’язки	Зв’язки
Реалізує логічні операції щодо побудови тональностей, генерації тональностей, перевірку відповідей, що приходять від Tonality Builder Controller.	Scale, Random

Таблиця 2.6. – CRC-картка для класу «Audio Engine»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Відтворює звуки нот та послідовностей нот по черзі.	Note

Таблиця 2.7. – CRC-картка для класу «Scale»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Структуризація тональності як сутності	Note

Таблиця 2.8. – CRC-картка для класу «Tonality Builder Controller»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Опрацьовує запити користувача під час роботи з Tonality Builder, отримує результати логічних операцій від Tonality Builder класу та інтерпретує їх на графічному інтерфейсі, контролює режими	Tonality Builder, Audio Engine

Обов’язки	Зв’язки
тренування та інструментарію.	

Таблиця 2.9. – CRC-картка для класу «Note»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Структуризація ноти як сутності	відсутні

Таблиця 2.10. – CRC-картка для класу «Random»

Базовий клас	Похідні класи (нащадки)
відсутній	відсутні
Обов’язки	Зв’язки
Рандомізація чисельних значень, що використовуються під час генерації у інших функціоналах програми	відсутні

2.2.1.4. Побудова об’єктної моделі (діаграми класів)

Як відомо, зазвичай об’єктна модель будується та демонструється через діаграму класів. Така діаграма класів позначає різноманітні об’єктні взаємозв’язки. Окрім цього, на діаграмі класів можна розгледіти як саме було виконано структуризацію класів. Серед цього можна відмітити види доступу до класових атрибутів та методів (приватний, публічний, або захищений), які позначаються відмітками +, - та #. Такі види доступу досягаються на практиці приписуванням у коді специфікаторів public, private, protected.

Важливо відмітити, що на діаграмі класів зазвичай вказуються різноманітні зв’язки між класами. Таких зв’язків на мові UML існує декілька, а саме серед них виділяють: композицію, агрегацію, асоціацію, залежність, реалізацію, спаадкування.

Узагальнити моделювання допоможе побудована діаграма класів (див рис.2.2.).

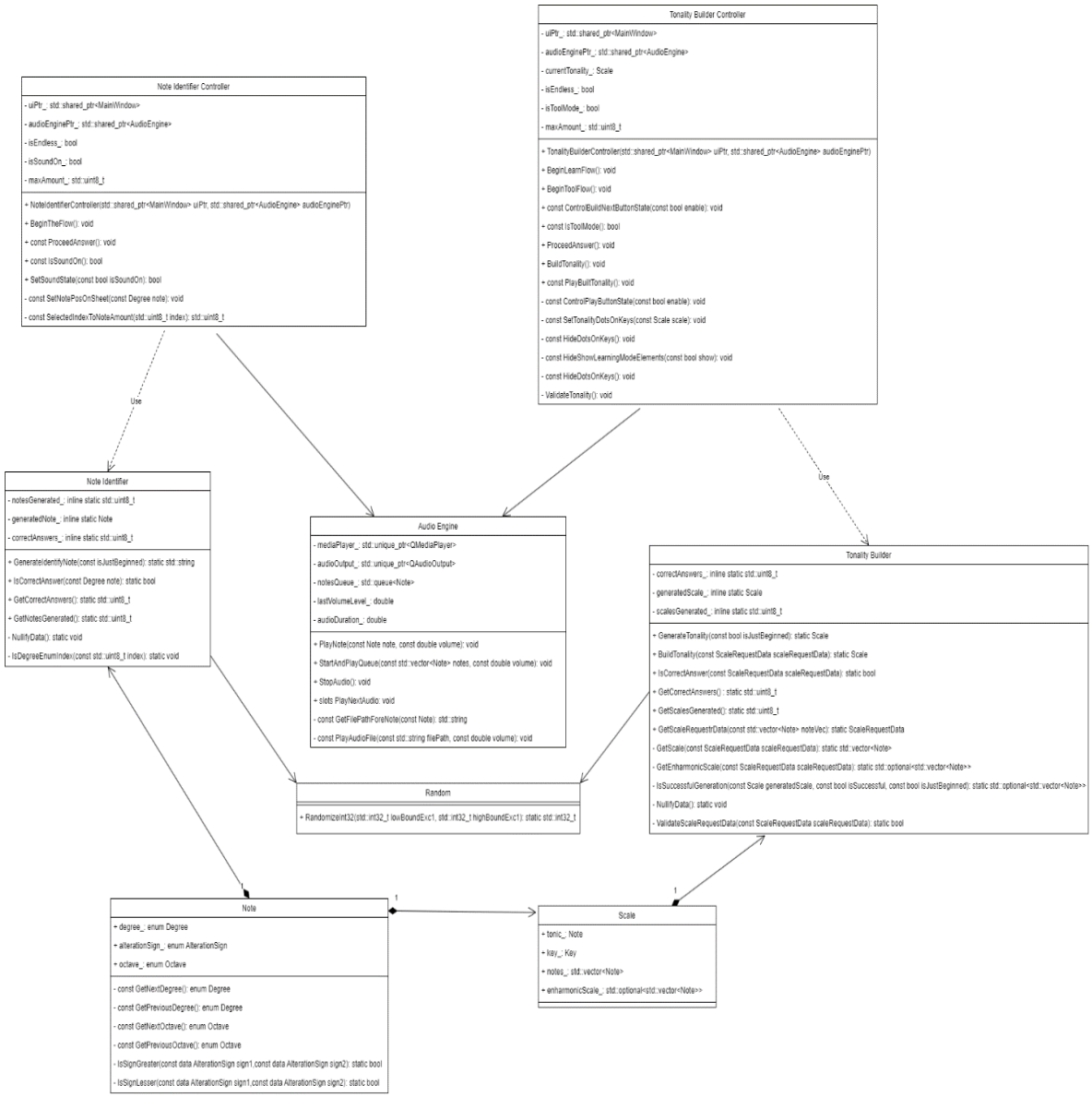

Рисунок 2.2. Діаграма класів

Класи пов’язані асоціацією: Note Identifier Controller та Audio Engine, Note Identifier та Random, Audio Engine та Note, Tonality Builder та Random, Tonality Builder Controller та Audio Engine.

Класи пов’язані композицією: Note Identifier та Note, Tonality Builder та Scale, Scale та Note.

Класи пов’язані реалізацією: Note Identifier Controller та Note Identifier, Tonality Builder Controller та Tonality Builder.

2.2.2. Проектування інтерфейсу користувача

2.2.2.1. Створення ескізів форм

Спираючись на те, що проектування інтерфейсу користувача та створення ескізів форм було зроблено студентом Максимом Лисенко у цій роботі, їх можна побачити у його роботі у пункті «2.2.2.1. Створення ескізів форм».

2.2.3. Проектування динаміки системи

У ролі об’єкту виступає користувач програмного забезпечення, який ініціює роботу класів через графічний інтерфейс. Самі звернення до інтерфейсу пропускаються, так як ця частина відноситься до роботи Лисенка Максима (див. дипломну роботу Лисенка Максима, пункт «2.2.3. Проектування динаміки системи). Розглядаються запити безпосередньо від контроллерів, через які здійснюється взаємодія між інтерфейсом там основною логічною складовою програми.

Опираючись на попередньо створену діаграму прецедентів [пункт 2.1.5], було розроблено дві діаграми послідовності для двох варіантів використання:

– Генерація випадкових нот на нотному стані для тренування (див. рис. 2.3)
– Генерація випадкової тональності для тренування (див. рис. 2.4)

Ці два прецеденти було обрано в зв’язку із тим, що найбільше логічної нагрузки на систему покладено саме у ціх прецедентах. «Побудова тональності за наданими користувачем даними» не була розглянута детально, так як практично виконує ті самі завдання, що й для генерації, але лише будує тональність за попередньо наданими даними.

На діаграмах не було ураховано сутності Scale та Note з тої причини, що вони виконують роль носїв інформації. Лише сутність Note має невелику кількість методів, що використовуються під час роботи застосунка, але ці методи виконують роль імітації операторів інкрементування, декрементування, порівняння «більше» та порівняння «менше». Сама сутність Scale повністю вміщує лише інформацію про тональність, не маючи жодних

методів що могли б виконувати інкрементування або декрементування, операції порівняння «більше» та «менше», як це є у сутності Note. Окрім того присутній перелік ступеней, знаків альтерації та октав. Усі ці перелічення прив’язані до певних класів, тому вони також не розглядаються.

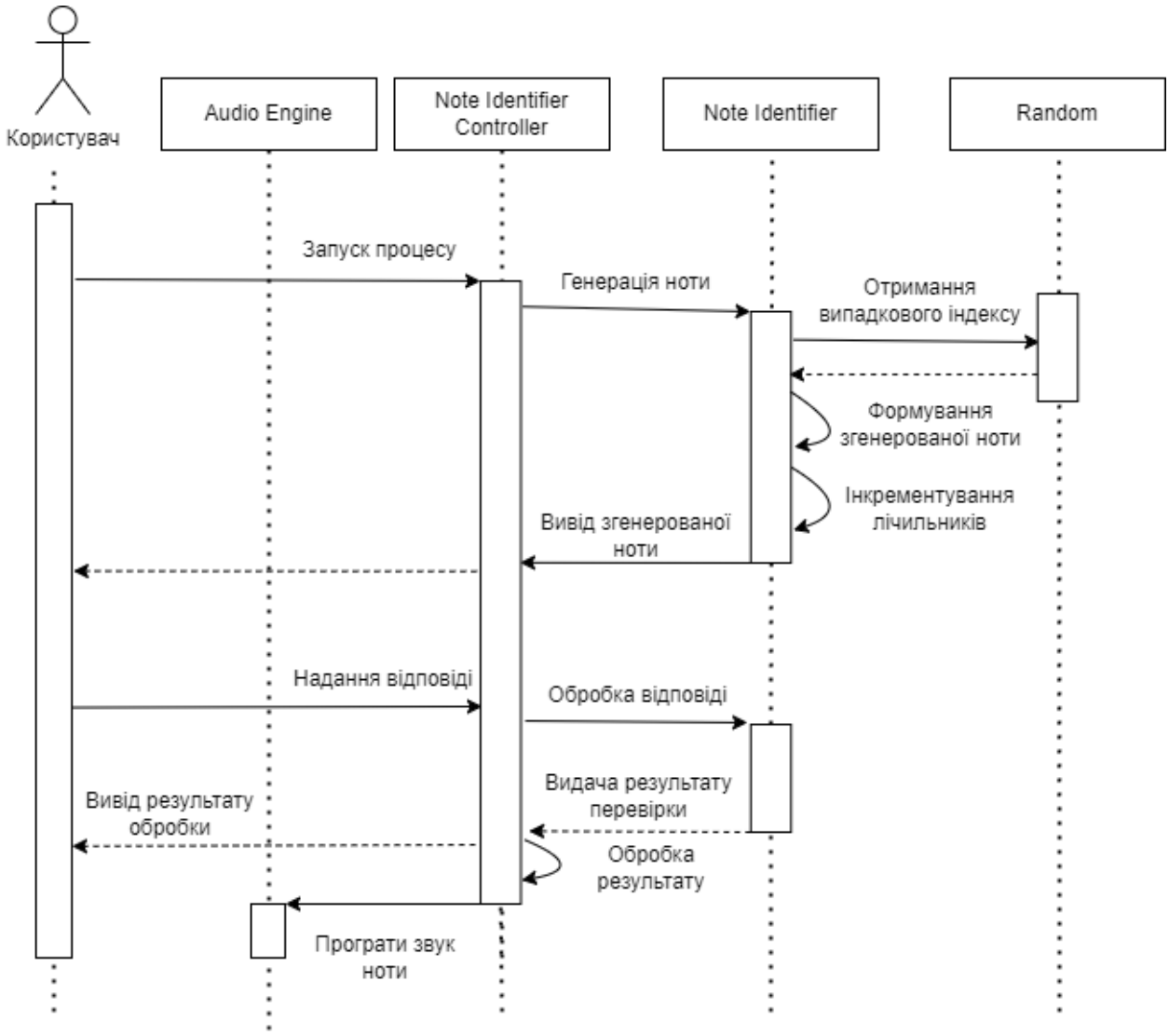

Рисунок 2.3. Діаграма послідовності варіанту використання «Генерація випадкових нот на нотному стані для тренування»

Сценарій з рис. 2.3 пояснює наступні дії:

– Користувач посилає запит на запуск процесу тренажеру ідентифікатора нот;
– Запит оброблюється контролером та посилається до основного логічного модуля ідентифікатору нот;

– Логічний модуль звертається до рандомізатора з метою згенерувати індекс ноти, що буде згенеровано для користувача;
– За отриманим індексом формується нота, інкрементуються лічильники щодо згенерованої кількості нот;
– Контролер отримує відповідь від логічного модулю, формує її на екрані та передає на форму;
– Паралельно із цим програється звук ноти для користувача.

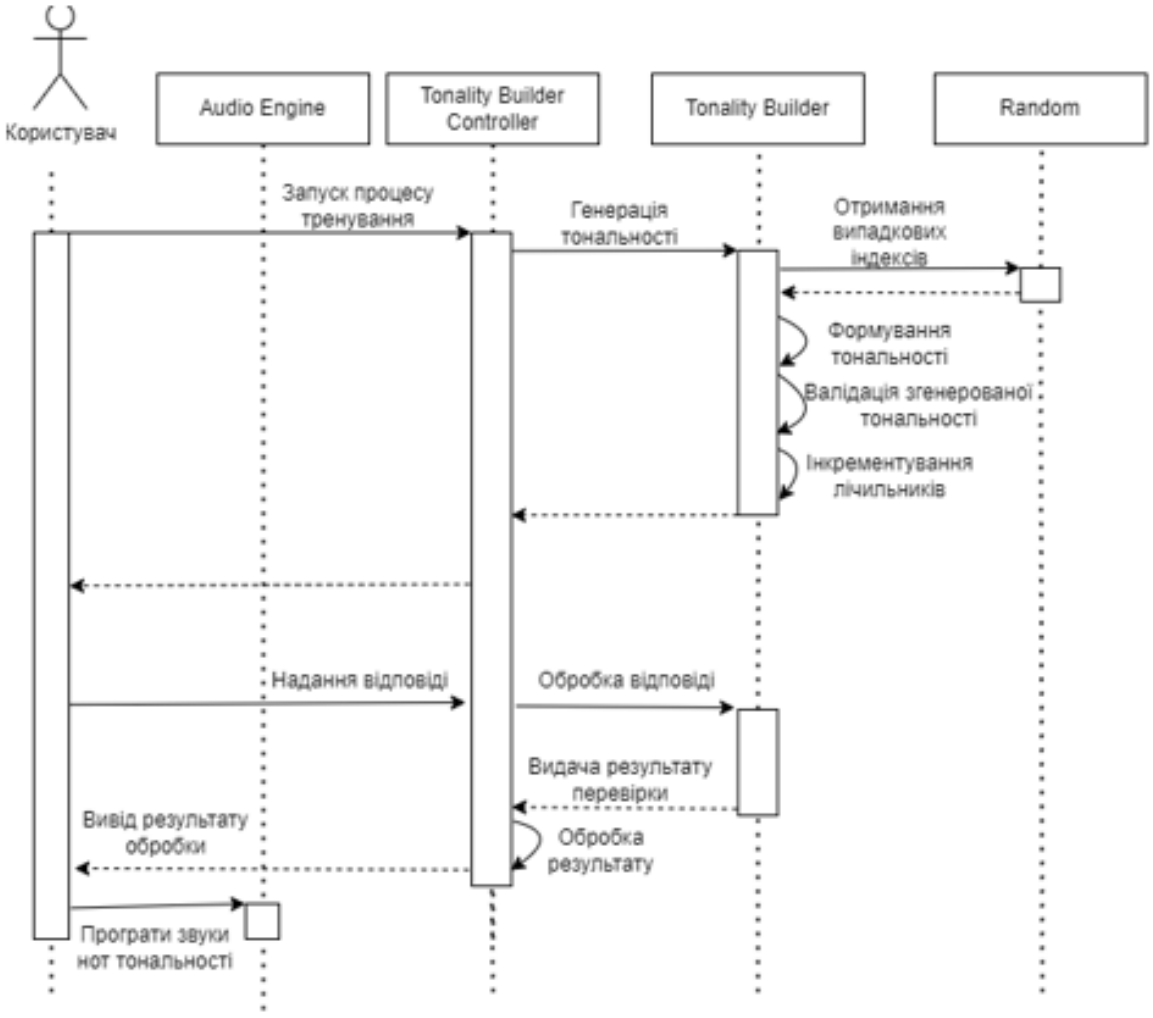

Рисунок 2.4. Діаграма послідовності варіанту використання «Генерація випадкових тональностей для тренування»

Сценарій з рис. 2.4 пояснює наступні дії:

– Користувач посилає запит на запуск процесу тренажеру будувальника тональності;

- Запит оброблюється контролером та посилається до основного логічного модуля будувальника тональності;
- Логічний модуль звертається до рандомізатора з метою згенерувати індекси тональності (ступінь, знак альтерації, лад), за якими буде згенеровано саму тональність для користувача;
- За отриманими індексами формується тональність, яка надалі буде валідуватися перед видачею як відповідь на запит. Інкрементуються лічильники щодо згенерованої кількості тональностей;
- Контролер отримує відповідь від логічного модулю, формує її на екрані та передає на форму;
- За необхідності, користувач може програти звуки нот тональності.

2.2.4. Вибір мови програмування та засобів спільної розробки

Для розробки продукту було вирішено використати мову програмування C++ у силу того, що дана мова дозволяє оперативно працювати із пам’яттю та надає гнучкість у розробці. Важливою складовою даної мови також є можливість програмувати за принципами ООП. Як відомо, ці принципи складають інкапсуляція, поліморфізм та наслідування, що C++ також має. Саме завдяки цім складовим розробка програми виявилась найбільш комфортною та ефективною.

Окрім самої мови програмування продукту, також застосовано автоматизовану систему збірки CMake. CMake слугує гнучкою системою збірки, завдяки якій можна розподіляти запускаємі частини програми на таргети – тобто об’єкти, які працюють по різному. Це є неймовірно корисним, як, наприклад, для багатосторонніх додатків на кшталт серверів. У випадку з даним програмним продуктом, тут застосовується два таргети – таргет «програма» та таргет «юніт тести». У першому запускається сама програма, у другому – юніт тести до неї. Але на практиці запуск усієї програми відбувається з інтерфейсної частини фреймворку QT, що під’єднаний до логічної частини додатку.

Проект використовує додаткові бібліотеки, що встановлює пакетний менеджер Conan. Завдяки цьому пакетному менеджеру можна вільно встановити необхідні для проекту бібліотеки та підключати їх як модулі до необхідних частин коду. Conan завантажує специфіковані у файлі conanfile.txt бібліотеки та під'єднує їх до репозиторію.

Деякі компоненти було встановлено завдяки використанню MSYS2 – емулятора терміналу Unix-подібних систем, що працює на Windows. Він дозволяє оперувати у системі як наче це робиться у Linux, що дозволяє ставити різні бібліотеки, які зазвичай вільно постачаються для Linux. Серед таких слід відмітити Python бібліотеки, що необхідні для роботи пакетного менеджеру Conan.

Необхідно відмітити, що з боку інтерфейсної частини тут застосовано фреймворк QT. Цей фреймворк було обрано з тої причини, що він чудово співпрацює із мовою C++, так як безпосередньо написаний на ній. Надважливо також сказати, що цей фреймворк забезпечує роботу на різних платформах, таких як Windows, Linux та iOS. Це робить розробку продуктів на цьому фреймворці гнучкою, адже забезпечує кросплатформенність додатку. Фреймворк дозволяє працювати із мультимедіа, надаючи власні бібліотеки для цього, серед яких було застосовано QMultimedia.

Ключовим для проекту елементом є використання системи контролю версій Git. Дана система дозволяє зберігати виконану роботу у гілках – свого роду контрольних точках, на яких можна було продовжити роботу у інший час, якщо з'явилась більш термінова задача, або передати роботу іншому співучаснику на доопрацювання, або навіть проводити спільну роботу на одній і тій самій гілці, майже забезпечуючи спільну роботу у реальному часі. Це дозволяє не перекидувати кожного разу одне одному одні і ті самі файли під час виконання роботи над кодом засобами електронної пошти або іншими засобами, а також дозволяє постійно мати актуальний стан репозиторію, шляхом додавання власної роботи до спільного репозиторію та контролю над прогрессом.

Проектний контроль та керування подальшою співпрацею по проекту відбувалась із використанням Jira та хмарного репозиторію BitBucket. Це було зроблено задля забезпечення проекту підтримки актуальності різних проблемних моментів, багів, та подальших розробок різних функціоналів. Таким чином, у Jira – програмному забезпеченні відстеження задач, було створено безліч «тікетів», тобто задач, на які ми орієнтувалися під час спільної роботи над проектом. Завдяки цьому, ми могли змогу завжди бачити наші ключові задачі, мати актуальну необхідну інформацію по тікетам, а також відстежувати ті тікети, які є завершеними, та ті, які ще необхідно зробити. Jira також дозволяє розподіляти тікети не тільки між співучасниками проекту, а ще й відокремлювати тікети по спрінтам – короктим проміжкам часу , за які необхідно зробити певний обсяг робіт по проекту. Це надає можливість грамотно розподілити порядок задач та ефективно спланувати дії.

Використовуючи BitBucket, ми могли зберігати наш репозиторій по проекту у хмарному сховищі та завжди мати доступ до нього у випадку технічних несправностей на наших персональних комп’ютерах, таким чином нічого не втративши. Окрім цього, користуючись ресурсами BitBucket ми могли змогу проводити код-рев’ю по зробленим тікетам, створюючи пул-реквести – запити на додавання нового коду до репозиторію по певному тікету. Це могли бути або новий функціонал, або баг-фікс та інше.

Висновки до пункту 2

За отриманими проектними рішеннями можна розробляти програмний код для реалізації застосунку.

Найбільш ключовим рішенням, особливо як для групової (парної) роботи слід назвати застосування системи контролю версій Git, а також програмне забезпечення Jira, завдяки якому уся робота контролюється та чітко відображена у реальному часі. Саме із подібним підходом можна говорити про грамотну організацію роботи над проектом.

Насамперед, важливо зазначити те, що саме по собі проектне рішення має багато шляхів розвитку, так як сама програма не є фінальною версією, і має ще

безліч ідей по реалізації різних аспектів музики – квінтове коло, розрішення акордів та нот та інших. Зокрема слід зазначити ще й можливість додавання теоретичного матеріалу прямо у додаток, що може допомогти користувачу зрозуміти всю суть музичної теорії напряму у програмі.

Також, після проектування та перших реалізацій проекту, з’являлось багато зауважень щодо власної роботи, а саме необхідність у рефакторінгу коду та деяких реструктурізацій коду, що можливо зробити у майбутньому. Ці зауваження були занесені до системи Jira, аби не загубити їх.

3. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

3.1. Тестування методом «білої скриньки»

Дана робота була виконана більше практично по бек-енд частині – створення логічних модулів, їх налагодження. Для тестування функціоналу було застосовано підхід використання юніт-тестів, що детально тестують кожен модуль окремо. Усього система налічує 20 одиниць юніт-тестів, з них – 3 тести створено студентом Лисенко Максимом (див. дипломну роботу студента Максима Лисенко, розділ «3. Тестування та налагодження»).

Серед юніт тестів виділяються наступні, подані нижче у таблиці 3.1.

Таблиця 3.1. – Перелік юніт тестів логічних модулів

Назва тестового випадку	Короткий опис	Очікуваний результат	Статус
Обробка коректних нот (ElegiaNotesIdentifierTest.CorrectNote)	Згенеровані ноти співпадають із запропонованими користувачем звідповідями	Функція перевірки повертає true	Пройдено
Обробка некоректних нот (ElegiaNotesIdentifierTest.WrongNote)	Згенерована нота не співпадає із запропоновою користувачем відповіддю	Функція перевірки повертає false	Пройдено
Перевірка гарантії неповторності ноти підряд (ElegiaNotesIdentifierTest.WrongNote)	Зі 100 згенерованих нот	Кожні дві підряд не будуть повторюватись	Пройдено
Побудування тональності за	При побудові тональності	Побудована тональність	Пройдено

Продовження таблиці 3.1.

параметрами тоніки, знака альтерації та ладу.		відповідає дійсній тональності	
Генерація тональності і її вгадуванян з вірною відповіддю	Згнерована тональність співпадає із запропонованою користувачем відповіддю	Функція перевірки повертає true	Пройдено
Генерація тональності і її невгадування з невірною відповіддю	Згнерована тональність не співпадає із запропонованою користувачем відповіддю	Функція перевірки повертає false	Пройдено
Ввід енгармонічної тональності після генерації	Коли вводиться енгармонічна тоніальність	Відповідь також вважається вірною	Пройдено
Перевірка гарантії неповторності тональності підряд	Зі 100 згенерованих тональностей	Кожні дві підряд не будуть повторюватись	Пройдено
Виконання конвертацій із Degree enum до std::string та навпаки	Виконується конвертація із enum/std::string	Успішно повертається конвертований std::string/enum	Пройдено
Виконання конвертацій із Octave	Виконується конвертація із	Успішно повертається	Пройдено

Продовження таблиці 3.1.

enum до std::string та навпаки	enum/std::string	конвертований std::string/enum	
Виконання конвертацій із AlterationSign enum до std::string та навпаки	Виконується конвертація із enum/std::string	Успішно повертається конвертований std::string/enum	Пройдено
Виконання конвертацій із Note enum до std::string та навпаки	Виконується конвертація із enum/std::string	Успішно повертається конвертований std::string/enum	Пройдено
Виконання конвертацій із Key enum до std::string та навпаки	Виконується конвертація із enum/std::string	Успішно повертається конвертований std::string/enum	Пройдено
Інкрементація структури Note	Викликана інкрементація operator++	Нота інкрементована корректно	Пройдено
Декрементація структури Note	Викликана декрементація operator--	Нота декрементована корректно	Пройдено
Порівняння структури Note	Викликані оператори порівняння operator> та operator<	Ноти порівнюються вірно	Пройдено
Рандомізація цілого 32-бітного числа	Викликана функція рандомізації 10 разів	Результат знаходиться у заданих межах	Пройдено

Тестування із використанням юніт-тестів показало надійність та коректність логічних процесів модулів бек-енду. Протестовано модулі функціоналу Note Identifier, Tonality Builder, Random, функції конвертації перечислень та перевантажені оператори Note.

3.2. Тестування методом «чорної скриньки»

Так як дана робота реалізована практично лише по частині бек-енду, тестування методом чорної скриньки не інформативне з точки зору виявлення помилок. Тестування методом чорної скриньки використовується для тестування взаємодії інтерфейсу та користувача і перевірки виконання основних функціональних складових (див. у дипломній роботі Лисенка Максима у пункті «3.2. Тестування методом чорної скриньки»).

Висновки до пункту 3

Невід’ємною частиною розробки програмних продуктів є їх тестування. На данному етапі, тестування було виконано шляхом написання юніт-тестів, а саме використовувались GTest (Google Test) тести, які допомагали протестувати продукт у різних сценаріях, зімітованих у самих тестах. Це дозволяє ще на етапі розробки виявити деякі баги та помилки під час розробки та ліквідувати їх.

ВИСНОВОК

Сьогоді є безліч засобів з навчання музичній теорії та практиці. Це є як мобільні додатки, так і веб-ресурси. Кожен з них надає допомогу людині у тому, аби отримати певні навички у музиці. Тобто, музична освіта не обмежується одними книжками та підручниками, що робить процес навчання більш цікавим та захоплюючим.

Під час розробки даного програмного забезпечення, було прораховано потреби користувача у тому, аби візуалізовувати музичні тональності на клавішах фортепіано. Водночас було пропрацьовано можливість прослуховувати тональність, а саме ноти що знаходяться у ній.

Те саме стосується і ідентифікатора нот. Вміння читати ноти з нотного стану на листку є необхідним для більшості музикантів, а вміння робити це швидко – ще важливіше. Відточити ці навички допоможе саме ідентифікатор, що не тільки вимальовує ноту на нотному стані, а ще й програє її звук, що допомагає тренувати абсолютний слух користувачу.

Після розробки, на думку спало багато ідей для розвитку проекту. Наприклад, ті ж сами музичні тональності, що візуально зображуються на віртуальному піаніно, можна було б також відображувати на інших інструментах: гітара, бас-гітара, флейта, скрипка та інші. Це розширило б коло музикантів, які користувалися б програмою та мали б за основний інструмент перелічені інструменти. Ще потрібно відмітити те, що можна реалізувати квінтове коле, що допомагає музиканту створювати музичні твори, знаючи, на які ступені можна переносити твір під час програвання, і що буде звучати гармонічно. В тому числі, можна додати теоретичну частину прямо у додаток, завдяки якій користувач матиме ще більш розгорнуту інформацію по музичній теорії.

ЛІТЕРАТУРА

1. І. В. Способін. Елементарна теорія музики / Державне музичне видавництво: М., 1963.
2. Пояснювальна записка до кваліфікаційної роботи бакалавра. Дніпро, Лисенко Максим, гр. 943.
3. QT Framework Documentation [Електронний ресурс] // Режим доступу до сайту: https://doc.qt.io/
4. Conan C++ Package Manager Documentation [Електронний ресурс] // Режим доступу до сайту: https://docs.conan.io/2/
5. MSYS2 Package Management Documentation [Електронний ресурс] // Режим доступу до сайту: https://www.msys2.org/docs/package-management/
6. Boost C++ Library Documentation [Електронний ресурс] // Режим доступу до сайту: https://www.boost.org/doc/libs/
7. FMT C++ Library API [Електронний ресурс] // Режим доступу до сайту: https://fmt.dev/latest/api.html
8. Lowinsky E. Tonality and atonality in sixteenth-century music. Berkeley, 1961.

ДОДАТКИ

ТЕКСТ ПРОГРАММИ

backend/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(ElegiaBE VERSION 0.1.0)

if (NOT EXISTS BACKEND_FOLDER_PATH)
   set(BACKEND_FOLDER_PATH
"${CMAKE_CURRENT_SOURCE_DIR}" CACHE PATH
"Location of backend folder")
   message("Backend folder path is:
${BACKEND_FOLDER_PATH}")
endif()

list(APPEND CMAKE_MODULE_PATH
"${CMAKE_CURRENT_SOURCE_DIR}/cmake")
# Conan package manager (for libraries like fmt, boost, etc...)
set (CONAN_CMAKE_FILE_PATH
"${BACKEND_FOLDER_PATH}/cmake/conan.cmake")
message("Looking for conan.cmake file in
${CONAN_CMAKE_FILE_PATH} ...")

# GTest
include(GoogleTest)
include(${CONAN_CMAKE_FILE_PATH})

# Downloaded Conan libs
conan_cmake_run(CONANFILE conanfile.txt BASIC_SETUP
CMAKE_TARGETS BUILD missing PROFILE default)
include(${BACKEND_FOLDER_PATH}/build/conanbuildinfo.c
make)
conan_basic_setup()

# Tests enable variable declaration
option(ENABLE_TESTING "Unit tests building/configuration"
ON)

# CMake GTest enabling if needed
if (ENABLE_TESTING)
   add_definitions(-DENABLE_TESTING)
endif ()

add_subdirectory(utils)
add_subdirectory(data)
add_subdirectory(elegia)
add_subdirectory(app)

if (ENABLE_TESTING)
   include(CTest)
   enable_testing()
   add_subdirectory(test)
endif()
```

backend/utils/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(utils LANGUAGES CXX)

set(${PROJECT_NAME}_HEADERS
   include/Random.h
)

set(${PROJECT_NAME}_SOURCES
   src/Random.cpp
)

set(${PROJECT_NAME}_MISC
   src/pch.h
)

add_library(${PROJECT_NAME} STATIC
   ${${PROJECT_NAME}_SOURCES}
   ${${PROJECT_NAME}_HEADERS}
   ${${PROJECT_NAME}_MISC}
)

add_library(elegia::utils ALIAS ${PROJECT_NAME})

target_compile_features(${PROJECT_NAME} PUBLIC
cxx_std_17)

target_include_directories(${PROJECT_NAME}
   PRIVATE
      ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
      ${CMAKE_CURRENT_SOURCE_DIR}/src
   PUBLIC
      ${CMAKE_CURRENT_SOURCE_DIR}/include
)

target_link_libraries(${PROJECT_NAME}
   PUBLIC
      ${CONAN_LIBS}
)

# Precompiled header
target_precompile_headers(${PROJECT_NAME} PRIVATE
src/pch.h)
```

backend/test/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(test LANGUAGES CXX)

set(SOURCES
   ElegiaNotesIdentifierTest.cpp
   ElegiaScaleBuilderTest.cpp
   ElegiaTranposerTest.cpp
   EnumConversionTest.cpp
   NoteTest.cpp
   RandomTest.cpp
)

set(MISC
   pch.h
)

include(FetchContent)

FetchContent_Declare(
   googletest
   GIT_REPOSITORY https://github.com/google/googletest.git
   GIT_TAG        release-1.11.0
)

FetchContent_MakeAvailable(googletest)

add_library(GTest::GTest INTERFACE IMPORTED)

# Prevent overriding the parent project's compiler/linker
# settings on Windows
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)

target_link_libraries(GTest::GTest INTERFACE gtest_main
gmock gmock_main)

add_executable(unit_tests
   ${MISC}
   ${SOURCES}
   ${UTILS}
)

target_include_directories(unit_tests
   PRIVATE
      ${CMAKE_CURRENT_SOURCE_DIR}
)

target_include_directories(unit_tests
```

```
    PUBLIC
        ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
)

target_link_libraries(unit_tests
    PRIVATE
        GTest::GTest
        elegia::utils
        elegia::data
        elegia::elegia
)

target_precompile_headers(unit_tests PRIVATE pch.h)

gtest_discover_tests(unit_tests WORKING_DIRECTORY
${CMAKE_RUNTIME_OUTPUT_DIRECTORY})
```

backend/elegia/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(elegia LANGUAGES CXX)

set(${PROJECT_NAME}_HEADERS
    include/NotesIdentifier.h
    include/ScaleBuilder.h
    include/Transposer.h
)

set(${PROJECT_NAME}_SOURCES
    src/NotesIdentifier.cpp
    src/ScaleBuilder.cpp
    src/Transposer.cpp
)

set(${PROJECT_NAME}_MISC
    src/pch.h
)

add_library(${PROJECT_NAME} STATIC
    ${${PROJECT_NAME}_SOURCES}
    ${${PROJECT_NAME}_HEADERS}
    ${${PROJECT_NAME}_MISC}
)

add_library(elegia::elegia ALIAS ${PROJECT_NAME})

target_compile_features(${PROJECT_NAME} PUBLIC
cxx_std_17)

target_include_directories(${PROJECT_NAME}
    PRIVATE
        ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
        ${CMAKE_CURRENT_SOURCE_DIR}/src
    PUBLIC
        ${CMAKE_CURRENT_SOURCE_DIR}/include
)

target_link_libraries(${PROJECT_NAME}
    PUBLIC
        ${CONAN_LIBS}
        elegia::utils
        elegia::data
)

# Precompiled header
target_precompile_headers(${PROJECT_NAME} PRIVATE
src/pch.h)
```

data/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(data LANGUAGES CXX)

set(${PROJECT_NAME}_HEADERS
    include/Enums.h
    include/Note.h
    include/Scale.h
    include/Types.h
)

set(${PROJECT_NAME}_SOURCES
    src/Enums.cpp
    src/Note.cpp
)

set(${PROJECT_NAME}_MISC
    src/pch.h
)

add_library(${PROJECT_NAME} STATIC
    ${${PROJECT_NAME}_SOURCES}
    ${${PROJECT_NAME}_HEADERS}
    ${${PROJECT_NAME}_MISC}
)

add_library(elegia::data ALIAS ${PROJECT_NAME})

target_compile_features(${PROJECT_NAME} PUBLIC
cxx_std_17)

target_include_directories(${PROJECT_NAME}
    PUBLIC
        ${PROJECT_SOURCE_DIR}/include
    PRIVATE
        ${PROJECT_SOURCE_DIR}/include
        ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}
        ${CMAKE_CURRENT_SOURCE_DIR}/src
)

target_link_libraries(${PROJECT_NAME}
    PUBLIC
        ${CONAN_LIBS}
        elegia::utils
)

# Precompiled header
target_precompile_headers(${PROJECT_NAME} PRIVATE
src/pch.h)
```

backend/app/CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24.1)

project(app VERSION 0.1.0)

# TODO: if needed in future, uncode this and paste .h file
#set(${PROJECT_NAME}_HEADERS)

set(${PROJECT_NAME}_SOURCES
    src/Main.cpp
)

set(${PROJECT_NAME}_MISC
    src/pch.h
)

add_executable(${PROJECT_NAME}
    ${${PROJECT_NAME}_SOURCES}
    ${${PROJECT_NAME}_MISC}
)

target_compile_features(${PROJECT_NAME} PUBLIC
cxx_std_17)

target_link_libraries(${PROJECT_NAME}
    PUBLIC
        elegia::utils
        elegia::data
        elegia::elegia
)
```

```
#file(
#  COPY
${CMAKE_CURRENT_SOURCE_DIR}/config/app.exe
#  DESTINATION
${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/.
#  FILE_PERMISSIONS OWNER_READ OWNER_WRITE
OWNER_EXECUTE GROUP_READ GROUP_EXECUTE
WORLD_READ WORLD_EXECUTE
#)

configure_file(
   ${CMAKE_CURRENT_SOURCE_DIR}/config/app-
config.json
   ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/app-
config.json
   COPYONLY
)

message("Running app.exe in
${CMAKE_RUNTIME_OUTPUT_DIRECTORY}")

# Precompiled header
target_precompile_headers(${PROJECT_NAME} PRIVATE
src/pch.h)
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)

if (WIN32)
   project(ElegiaApp VERSION 0.1 LANGUAGES CXX)
elseif(UNIX)
   project(ElegiaApp)
endif()

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_PREFIX_PATH $ENV{QTDIR})

find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS
Core)
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS
Widgets)
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS
Multimedia)
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS
MultimediaWidgets)
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS
Gui)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED
COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED
COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED
COMPONENTS Multimedia)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED
COMPONENTS MultimediaWidgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED
COMPONENTS Gui)

set(QT_USE_QTMULTIMEDIA TRUE)
set(QT_USE_QTMULTIMEDIAWIDGETS TRUE)

add_subdirectory(backend)
set(CMAKE_CXX_FLAGS "-L
${BACKEND_FOLDER_PATH}/build/conanlibs")
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

message("Current Qt version is:
${QT_VERSION_MAJOR}.${QT_VERSION_MINOR}")

set(UI_DIR
   ui/mainwindow.ui
)

set(INCLUDE_H_DIR
   include/MainWindow.h
   include/NotesIdentifierController.h
   include/AudioEngine.h
   include/TonalityBuilderController.h
   include/TransposerController.h
)

set(SOURCES_CPP_DIR
   src/main.cpp
   src/MainWindow.cpp
   src/NotesIdentifierController.cpp
   src/AudioEngine.cpp
   src/TonalityBuilderController.cpp
   src/TransposerController.cpp
)

set(RESOURCES_DIR
   resources/Audios.qrc
   resources/Images.qrc
   resources/Styles.qrc
)

set(CMAKE_AUTOUIC_SEARCH_PATHS "ui")

if (WIN32)
   qt_add_executable(${PROJECT_NAME}
         WIN32
         MANUAL_FINALIZATION
         ${INCLUDE_H_DIR}
         ${SOURCES_CPP_DIR}
         ${UI_DIR}
         ${RESOURCES_DIR}
   )
   qt_finalize_executable(${PROJECT_NAME})
elseif(UNIX)
   qt_add_executable(${PROJECT_NAME}
         MANUAL_FINALIZATION
         ${INCLUDE_DIR}
         ${SOURCES_DIR}
         ${UI_DIR}
         ${RESOURCES_DIR}
   )
   qt_finalize_executable(${PROJECT_NAME})
endif()

target_compile_features(${PROJECT_NAME} PUBLIC
cxx_std_17)

include(${BACKEND_FOLDER_PATH}/build/conanbuildinfo.c
make)
conan_basic_setup()

target_link_libraries(${PROJECT_NAME}
   PUBLIC
      elegia::utils
      elegia::data
      elegia::elegia
   PRIVATE
      Qt${QT_VERSION_MAJOR}::Widgets
      Qt${QT_VERSION_MAJOR}::Core
      Qt${QT_VERSION_MAJOR}::Multimedia
      Qt${QT_VERSION_MAJOR}::Gui
)

# Precompiled header
target_precompile_headers(${PROJECT_NAME} PRIVATE
src/pch.h)
```

.gitignore

```
# This file is used to ignore files which are generated
# ----------------------------------------------------------------------------
backend
build
CMakeLists.txt.user
CMakeLists.txt.user.*
```

```
*~
*.autosave
*.a
*.core
*.moc
*.o
*.obj
*.orig
*.rej
*.so
*.so.*
*_pch.h.cpp
*_resource.rc
*.qm
.#*
*.*#
core
!core/
tags
.DS_Store
.directory
*.debug
Makefile*
*.prl
*.app
moc_*.cpp
ui_*.h
qrc_*.cpp
Thumbs.db
*.res
*.rc
/.qmake.cache
/.qmake.stash

# qtcreator generated files
*.pro.user*

# xemacs temporary files
*.flc

# Vim temporary files
.*.swp

# Visual Studio generated files
*.ib_pdb_index
*.idb
*.ilk
*.pdb
*.sln
*.suo
*.vcproj
*vcproj.*.*.user
*.ncb
*.sdf
*.opensdf
*.vcxproj
*vcxproj.*

# MinGW generated files
*.Debug
*.Release

# Python byte code
*.pyc

# Binaries
# --------
*.dll
*.exe
```

backend/.gitignore

```
# CMake build results.
.cmake
build/*
app/CMakeFiles
*/Makefile
Makefile
conan_toolchain.cmake
conan.lock
conanbuildinfo.cmake
conanbuildinfo.txt
conaninfo.txt
CTestTestfile.cmake
DartConfiguration.tcl
graph_info.json
*/CMakeFiles
CMakeFiles
cmake_install.cmake
*/CTestTestfile.cmake
Testing
CMakeCache.txt
bin
*.user
*.autosave
lib
unit_tests[1]_include.cmake
unit_tests[1]_tests.cmake

# GTest
_deps

# CMake user presets
CMakeUserPresets.json
CMakePresets.json

# VsCode
.VsCode
```

backend/conanfile.txt

```
[requires]
boost/1.71.0
fmt/7.0.3

[options]
boost:without_fiber=True
boost:without_stacktrace=True
boost:without_test=True
boost:shared=True
fmt:shared=True

[imports]
bin, *.dll -> ./conanlibs
lib, *.dll.a -> ./conanlibs
bin, *.a -> ./conanlibs
lib, *.a -> ./conanlibs

[generators]
cmake
```

backend/Main.cpp

```
#include "pch.h"

int main()
{
   std::cout << "Hello!";

   return 0;
}
```

backend/pch.h

```
#pragma once

#include <iostream>
#include <string>

#ifdef WIN32
#  pragma warning(push)
// warning C4100 : 'name' : unreferenced formal parameter.
#  pragma warning(disable : 4100)
#endif
```

```
#ifdef WIN32
#  pragma warning(pop)
#endif
```

backend/data/Enums.h

```
#pragma once

namespace data
{

struct Note;

//
// Enumerations
//

enum class Degree
{
   C = 1,        // Do        (до)
   D = 3,        // Re        (ре)
   E = 5,        // Mi        (ми)
   F = 6,        // Fa        (фа)
   G = 8,        // Sol       (соль)
   A = 10,       // La        (ля)
   B = 12        // Si        (си)
};

enum class Octave
{
   SUB_CONTRA   = 0,      // Субконтроктава   (C0)
   CONTRA       = 1,      // Контроктава      (C1)
   GREAT        = 2,      // Большая октава   (C2)
   SMALL        = 3,      // Малая октава     (C3)
   LINE1        = 4,      // Первая октава    (C4)
   LINE2        = 5,      // Вторая октава    (C5)
   LINE3        = 6,      // Третья октава    (C6)
   LINE4        = 7,      // Четвертая октава (C7)
   LINE5        = 8,      // Пятая октава     (C8)
   LINE6        = 9,      // Шестая октава    (C9)
};

enum class AlterationSign
{
   //! TODO: сверхважная тудушка. Возможно можно просто
использовать бекар вместо NONE
   // Основная ступень
   NONE       = 0,

   // Производные ступени
   SHARP        = 1,    // Диез
   FLAT         = 2,    // Бемоль

   // Дополнительные знаки альтерации
   DOUBLE_SHARP  = 3,   // Дубль Диез
   DOUBLE_FLAT   = 4, // Дубль Бемоль
   BEKAR        = 5 // Бекар
};

enum class Key
{
   MAJOR = 0,
   MINOR = 1
};


//
// Enums conversion to text (std::string)
//

std::int8_t AlterationSign2Int8(const data::AlterationSign altSign);
std::string AlterationSign2String(const data::AlterationSign
altSign);
std::string Degree2String(const data::Degree degree);
std::int8_t Degree2Int8(const data::Degree degree);
//! isFullyNamed - указывает на то, возвращать ли ноту вместе
с октавой, или просто без октавы.
//! Пример: Cb5 -> isFullyNamed (true)
//!         Cb  -> isFullyNamed (false)
std::string Key2String(const data::Key scale);
std::string Note2String(const data::Note& note, const bool
isFullyNamed);
std::string Octave2String(const data::Octave octave);

//
// Text (std::string) conversion to Enums
//

data::AlterationSign String2AlterationSign(const std::string& str);
data::Degree String2Degree(const std::string& str);
data::Key String2Key(const std::string& str);
data::Octave String2Octave(const std::string& str);

}   // namespace data
```

backend/data/Note.h

```
#pragma once

#include "Types.h"

namespace data
{

struct Note
{
   //! Ступень звукоряда.
   data::Degree degree_;
   //! Знак альтерации.
   data::AlterationSign alterationSign_ =
data::AlterationSign::NONE;
   //! Октава ступени.
   data::Octave octave_ = data::Octave::LINE1;

   //
   //  Private methods.
   //
private:
   //! Gets next degree after the current degree.
   Degree GetNextDegree() const;
   //! Gets previous degree before the current degree.
   Degree GetPreviousDegree() const;
   //! Gets next octave after the current octave.
   Octave GetNextOctave() const;
   //! Gets previous octave before the current octave.
   Octave GetPreviousOctave() const;

   //
   //  Так как перегрузка операторов enum class невозможна,
здесь созданы пару имплементаций ниже.
   //
   //! Checks whether first alteration sign is greater than second.
   static bool IsSignGreater(const data::AlterationSign& sign1,
const data::AlterationSign& sign2);
   //! Checks whether first alteration sign is lesser than second.
   static bool IsSignLesser(const data::AlterationSign& sign1,
const data::AlterationSign& sign2);

   //
   //  Overloaded operators.
   //
public:
   void operator=(const Note& note);
   bool operator==(const Note& note) const;
   bool operator>(const Note& note) const;
   bool operator<(const Note& note) const;
   Note& operator++();
   Note& operator--();
};


};   // namespace data
```

backend/data/Scale.h

```
#pragma once
```

```
#include "Types.h"
#include "Note.h"

namespace data
{

struct Scale
{
    //! Тоника - первая ступень тональности.
    Note tonic_;
    //! Наклонение (мажор, минор).
    Key key_;
    //! Ноты тональности.
    NoteVec notes_;
    //! Энгармоническая тональность данной тональности.
    NoteVecOpt enharmonicScale_;

    void operator=(const Scale& scale)
    {
        tonic_ = scale.tonic_;
        key_ = scale.key_;
        notes_ = scale.notes_;
        enharmonicScale_ = scale.enharmonicScale_;
    }

    bool operator!=(const Scale& scale) const
    {
        return !(
            key_ == scale.key_ && tonic_ == scale.tonic_ && notes_
== scale.notes_
            && enharmonicScale_ == scale.enharmonicScale_);
    }

    bool operator==(const Scale& scale) const
    {
        return key_ == scale.key_ && tonic_ == scale.tonic_ &&
notes_ == scale.notes_
            && enharmonicScale_ == scale.enharmonicScale_;
    }
};

struct ScaleRequestData
{
    //! Тоника - первая ступень тональности.
    Degree tonic_;
    //! Знак альтерации тоники.
    AlterationSign alterationSign_;
    //! Наклонение (мажор, минор).
    Key key_;

    bool operator==(const ScaleRequestData& scaleRequestData)
const
    {
        return (
            tonic_ == scaleRequestData.tonic_ && alterationSign_ ==
scaleRequestData.alterationSign_
            && key_ == scaleRequestData.key_);
    }

    bool operator>(const ScaleRequestData& scaleRequestData)
const
    {
        return (
            tonic_ > scaleRequestData.tonic_ && alterationSign_ >
scaleRequestData.alterationSign_
            && key_ > scaleRequestData.key_);
    }

    bool operator<(const ScaleRequestData& scaleRequestData)
const
    {
        return (
            tonic_ < scaleRequestData.tonic_ && alterationSign_ <
scaleRequestData.alterationSign_
            && key_ < scaleRequestData.key_);
    }
};

}   // namespace data
```

backend/data/Types.h

```
#pragma once

#include "Enums.h"

namespace data
{

//
//  Forward declarations
//

struct Note;
struct Scale;

//  Type aliases
using DoubleOpt = std::optional<std::double_t>;
using BoolOpt = std::optional<bool>;
using Int8Opt = std::optional<std::int8_t>;
using Int16Opt = std::optional<std::int16_t>;
using Int32Opt = std::optional<std::int32_t>;
using Int64Opt = std::optional<std::int64_t>;
using NoteVec = std::vector<Note>;
using NoteVecOpt = std::optional<NoteVec>;
using ScaleTuple = std::tuple<data::Degree, data::AlterationSign,
data::Key>;
using StringOpt = std::optional<std::string>;
using UInt8Opt = std::optional<std::uint8_t>;
using UInt16Opt = std::optional<std::uint16_t>;
using UInt32Opt = std::optional<std::uint32_t>;
using UInt64Opt = std::optional<std::uint64_t>;

}   // namespace data
```

backend/data/Enums.cpp

```
// clang-format off
#include "pch.h"

#include "Enums.h"
#include "Note.h"

namespace utils
{

template<typename L, typename R>
boost::bimap<L, R> MakeBimap(std::initializer_list<typename
boost::bimap<L, R>::value_type> list)
{
    return boost::bimap<L, R>(list.begin(), list.end());
}

}   // namespace utils

namespace data
{

namespace details
{

//
//  Enum-String bimaps definitions
//

using utils::MakeBimap;


const boost::bimap<data::AlterationSign, std::string>
AlterationSignBimap(
    MakeBimap<data::AlterationSign, std::string>({
        { data::AlterationSign::NONE,           std::string{ "" } },
        { data::AlterationSign::SHARP,          std::string{ "#" } },
        { data::AlterationSign::FLAT,           std::string{ "b" } },
```

```
        { data::AlterationSign::DOUBLE_SHARP,       std::string{
"##" } },
        { data::AlterationSign::DOUBLE_FLAT,        std::string{
"bb" } },
        { data::AlterationSign::BEKAR,              std::string{ "♮" } }
    })
);

const boost::bimap<data::AlterationSign, std::uint8_t>
AlterationSignBimapInt(
    MakeBimap<data::AlterationSign, std::uint8_t>({
        { data::AlterationSign::NONE,               std::uint8_t{ 0 } },
        { data::AlterationSign::SHARP,              std::uint8_t{ 1 } },
        { data::AlterationSign::FLAT,               std::uint8_t{ 2 } },
        { data::AlterationSign::DOUBLE_SHARP,       std::uint8_t{
3 } },
        { data::AlterationSign::DOUBLE_FLAT,        std::uint8_t{ 4
} },
        { data::AlterationSign::BEKAR,              std::uint8_t{ 5 } }
    })
);

const boost::bimap<data::Degree, std::string> DegreeBimap(
    MakeBimap<data::Degree, std::string>({
        { data::Degree::C,      std::string{ "C" } },
        { data::Degree::D,      std::string{ "D" } },
        { data::Degree::E,      std::string{ "E" } },
        { data::Degree::F,      std::string{ "F" } },
        { data::Degree::G,      std::string{ "G" } },
        { data::Degree::A,      std::string{ "A" } },
        { data::Degree::B,      std::string{ "B" } }
    })
);

const boost::bimap<data::Degree, std::uint8_t> DegreeBimapInt(
    MakeBimap<data::Degree, std::uint8_t>({
        { data::Degree::C,      std::uint8_t{ 1 } },
        { data::Degree::D,      std::uint8_t{ 3 } },
        { data::Degree::E,      std::uint8_t{ 5 } },
        { data::Degree::F,      std::uint8_t{ 6 } },
        { data::Degree::G,      std::uint8_t{ 8 } },
        { data::Degree::A,      std::uint8_t{ 10 } },
        { data::Degree::B,      std::uint8_t{ 12 } }
    })
);

const boost::bimap<data::Key, std::string> KeyBimap(
    MakeBimap<data::Key, std::string>({
        { data::Key::MAJOR, std::string{ "Major" }},
        { data::Key::MINOR, std::string{ "Minor" }}
    })
);

const boost::bimap<data::Octave, std::string> OctaveBimap(
    MakeBimap<data::Octave, std::string>({
        { data::Octave::SUB_CONTRA,     std::string{ "0" } },
        { data::Octave::CONTRA,         std::string{ "1" } },
        { data::Octave::GREAT,          std::string{ "2" } },
        { data::Octave::SMALL,          std::string{ "3" } },
        { data::Octave::LINE1,          std::string{ "4" } },
        { data::Octave::LINE2,          std::string{ "5" } },
        { data::Octave::LINE3,          std::string{ "6" } },
        { data::Octave::LINE4,          std::string{ "7" } },
        { data::Octave::LINE5,          std::string{ "8" } },
        { data::Octave::LINE6,          std::string{ "9" } }
    })
);

}   // namespace details

std::string Degree2String(const data::Degree degree)
{
    return details::DegreeBimap.left.at(degree);
}

std::int8_t Degree2Int8(const data::Degree degree)
{
    return details::DegreeBimapInt.left.at(degree);
}

std::string Key2String(const data::Key scale)
{
    return details::KeyBimap.left.at(scale);
}

std::string Note2String(const data::Note& note, const bool
isFullyNamed)
{
    if (isFullyNamed)
    {
        return { details::DegreeBimap.left.at(note.degree_) +
            details::AlterationSignBimap.left.at(note.alterationSign_)
+
            details::OctaveBimap.left.at(note.octave_) };
    }

    return { details::DegreeBimap.left.at(note.degree_) +
        details::AlterationSignBimap.left.at(note.alterationSign_) };
}

std::string Octave2String(const data::Octave octave)
{
    return details::OctaveBimap.left.at(octave);
}

data::AlterationSign String2AlterationSign(const std::string& str)
{
    try
    {
        return details::AlterationSignBimap.right.at(str);
    }
    catch(const std::exception& e)
    {
        throw std::runtime_error( fmt::format("Invalid value of
AlterationSign enum provided - {}", str));
    }

}

data::Degree String2Degree(const std::string& str)
{
    try
    {
        return details::DegreeBimap.right.at(str);
    }
    catch (const std::exception& ex)
    {
        throw std::runtime_error( fmt::format("Invalid value of
Degree enum provided - {}", str));
    }
}

data::Key String2Key(const std::string& str)
{
    try
    {
        return details::KeyBimap.right.at(str);
    }
    catch (const std::exception& ex)
    {
        throw std::runtime_error( fmt::format("Invalid value of Key
enum provided - {}", str));
    }
}

data::Octave String2Octave(const std::string& str)
{
    try
    {
        return details::OctaveBimap.right.at(str);
    }
    catch (const std::exception& ex)
    {
        throw std::runtime_error( fmt::format("Invalid value of
Octave enum provided - {}", str));
    }
```

```
}

std::string AlterationSign2String(const data::AlterationSign
altSign)
{
   return details::AlterationSignBimap.left.at(altSign);
}

std::int8_t AlterationSign2Int8(const data::AlterationSign altSign)
{
   return details::AlterationSignBimapInt.left.at(altSign);
}

}  // namespace data
// clang-format on
```

backend/data/Note.cpp

```
#include "Note.h"

namespace details
{

static const std::vector<data::Degree> DegreeSequence{
   data::Degree::C, data::Degree::D, data::Degree::E,
data::Degree::F,
   data::Degree::G, data::Degree::A, data::Degree::B,
};

static const std::vector<data::Octave> OctaveSequence{
   data::Octave::SUB_CONTRA, data::Octave::CONTRA,
data::Octave::GREAT, data::Octave::SMALL,
   data::Octave::LINE1,     data::Octave::LINE2,
   data::Octave::LINE3, data::Octave::LINE4,
   data::Octave::LINE5,     data::Octave::LINE6,
};

}   // namespace details

namespace data
{

Degree Note::GetNextDegree() const
{
   if (this->octave_ == data::Octave::LINE6)
   {
      return this->degree_;
   }

   if (this->degree_ == data::Degree::B)
   {
      return data::Degree::C;
   }

   for (int i = 0; i < details::DegreeSequence.size(); i++)
   {
      if (details::DegreeSequence[i] == this->degree_)
      {
         return details::DegreeSequence[++i];
      }
   }

   // Никогда не должно происходить
   throw std::runtime_error("Failed to get next degree.");
}

Degree Note::GetPreviousDegree() const
{
   if (this->octave_ == data::Octave::SUB_CONTRA)
   {
      return this->degree_;
   }

   if (this->degree_ == data::Degree::C)
   {
      return data::Degree::B;
   }

   for (int i = 0; i < details::DegreeSequence.size(); i++)
   {
      if (details::DegreeSequence[i] == this->degree_)
      {
         return details::DegreeSequence[--i];
      }
   }

   // Никогда не должно происходить
   throw std::runtime_error("Failed to get previous degree.");
}

Octave Note::GetNextOctave() const
{
   if (this->degree_ == data::Degree::B && this->octave_ ==
data::Octave::LINE6)
   {
      return this->octave_;
   }

   for (int i = 0; i < details::OctaveSequence.size(); i++)
   {
      if (details::OctaveSequence[i] == this->octave_)
      {
         return details::OctaveSequence[++i];
      }
   }

   // Никогда не должно происходить
   throw std::runtime_error("Failed to get next octave.");
}

Octave Note::GetPreviousOctave() const
{
   if (this->degree_ == data::Degree::C && this->octave_ ==
data::Octave::SUB_CONTRA)
   {
      return this->octave_;
   }

   for (int i = 0; i < details::OctaveSequence.size(); i++)
   {
      if (details::OctaveSequence[i] == this->octave_)
      {
         return details::OctaveSequence[--i];
      }
   }

   // Никогда не должно происходить
   throw std::runtime_error("Failed to get previous degree.");
}

bool Note::IsSignGreater(const data::AlterationSign& sign1, const
data::AlterationSign& sign2)
{
   if (sign1 != sign2)
   {
      // Первый знак - дубль диез.
      if (sign1 == AlterationSign::DOUBLE_SHARP)
      {
         return true;
      }
      // Первый знак - диез.
      if (sign1 == AlterationSign::SHARP && sign2 !=
AlterationSign::DOUBLE_SHARP)
      {
         return true;
      }
      //! Первый знак - бекар или ничего.
      //! TODO: кстати, знак "ничего" стоит сменить на бекар.
      if ((sign1 == AlterationSign::BEKAR || sign1 ==
AlterationSign::NONE)
         && sign2 != AlterationSign::DOUBLE_SHARP && sign2
!= AlterationSign::SHARP)
      {
         return true;
      }
      // Первый знак - бемоль.
```

```
        if (sign1 == AlterationSign::FLAT && sign2 !=
AlterationSign::DOUBLE_FLAT)
        {
            return false;
        }
        // Первый знак - дубль бемоль.
        if (sign1 == AlterationSign::DOUBLE_FLAT)
        {
            return false;
        }
    }

    // Знаки одинаковые.
    return false;
}

bool Note::IsSignLesser(const data::AlterationSign& sign1, const
data::AlterationSign& sign2)
{
    if (sign1 != sign2)
    {
        // Второй знак - дубль диез.
        if (sign2 == AlterationSign::DOUBLE_SHARP)
        {
            return true;
        }
        // Второй знак - диез.
        if (sign2 == AlterationSign::SHARP && sign1 !=
AlterationSign::DOUBLE_SHARP)
        {
            return true;
        }
        //! Второй знак - бекар или ничего.
        //! TODO: кстати, знак "ничего" стоит сменить на бекар.
        if ((sign2 == AlterationSign::BEKAR || sign2 ==
AlterationSign::NONE)
            && sign1 != AlterationSign::DOUBLE_SHARP && sign1
!= AlterationSign::SHARP)
        {
            return true;
        }
        // Второй знак - бемоль.
        if (sign2 == AlterationSign::FLAT && sign1 !=
AlterationSign::DOUBLE_FLAT)
        {
            return false;
        }
        // Второй знак - дубль бемоль.
        if (sign2 == AlterationSign::DOUBLE_FLAT)
        {
            return false;
        }
    }

    // Знаки одинаковые.
    return false;
}

void Note::operator=(const Note& note)
{
    degree_ = note.degree_;
    alterationSign_ = note.alterationSign_;
    octave_ = note.octave_;
}

bool Note::operator==(const Note& note) const
{
    return degree_ == note.degree_ && octave_ == note.octave_
        && alterationSign_ == note.alterationSign_;
}

bool Note::operator>(const Note& note) const
{
    return (degree_ >= note.degree_ &&
IsSignGreater(alterationSign_, note.alterationSign_))
        || (degree_ > note.degree_ && octave_ >= note.octave_) ||
octave_ > note.octave_;
}

bool Note::operator<(const Note& note) const
{
    return (degree_ <= note.degree_ &&
IsSignLesser(alterationSign_, note.alterationSign_))
        || (degree_ < note.degree_ && octave_ <= note.octave_) ||
octave_ < note.octave_;
}

Note& Note::operator++()
{
    switch (this->alterationSign_)
    {
    case data::AlterationSign::FLAT:
    {
        this->alterationSign_ = data::AlterationSign::NONE;
        break;
    }
    case data::AlterationSign::NONE:
    {
        //! TODO: Возможно не помешало бы сделать резолвер
для E#/Eb и B#/Bb
        if (this->degree_ == data::Degree::E)
        {
            this->degree_ = GetNextDegree();
        }
        else if (this->degree_ == data::Degree::B)
        {
            this->degree_ = GetNextDegree();
            this->octave_ = GetNextOctave();
        }
        else
        {
            this->alterationSign_ = data::AlterationSign::SHARP;
        }

        break;
    }
    //! TODO: в будущем стоит рассмотреть ::DOUBLE_SHARP
и ::DOUBLE_FLAT
    case data::AlterationSign::SHARP:
    {
        if (this->degree_ != data::Degree::B && this->degree_ !=
data::Degree::E)
        {
            this->alterationSign_ = data::AlterationSign::NONE;
        }

        this->degree_ = GetNextDegree();
        if (this->degree_ == data::Degree::C)
        {
            this->octave_ = GetNextOctave();
        }
    }
    }

    return *this;
}

Note& Note::operator--()
{
    switch (this->alterationSign_)
    {
    //! TODO: в будущем стоит рассмотреть ::DOUBLE_SHARP
и ::DOUBLE_FLAT
    case data::AlterationSign::FLAT:
    {
        if (this->degree_ != data::Degree::F)
        {
            this->alterationSign_ = data::AlterationSign::NONE;
        }

        this->degree_ = GetPreviousDegree();
        if (this->degree_ == data::Degree::B)
        {
            this->octave_ = GetPreviousOctave();
        }
        break;
```

```
    }
    case data::AlterationSign::NONE:
    {
        //! TODO: Возможно не помешало бы сделать резолвер
для E#/Eb и B#/Bb
        if (this->degree_ == data::Degree::E)
        {
            this->degree_ = GetPreviousDegree();
        }
        else if (this->degree_ == data::Degree::C)
        {
            this->degree_ = GetPreviousDegree();
            this->octave_ = GetPreviousOctave();
        }
        else
        {
            this->alterationSign_ = data::AlterationSign::FLAT;
        }
        break;
    }
    case data::AlterationSign::SHARP:
    {
        this->alterationSign_ = data::AlterationSign::NONE;
    }
    }

    return *this;
}

}   // namespace data
```

backend/data/pch.h

```
#pragma once

//
//  STL
//

#include <string>
#include <vector>

//
//  Boost
//

#include <boost/bimap.hpp>

//
//  FMT
//
#define FMT_HEADER_ONLY
#include <fmt/format.h>
```

backend/elegia/NotesIdentifier.h

```
#pragma once

namespace data
{
enum class Degree;
}

namespace elegia
{

//
//  Класс для работы определителя нот.
//  Генерирует ноты для определения, обрабатывает ответы
пользователя.
//

class NotesIdentifier
{
    //
    //  Public interface.
    //
public:
    //! Генерирует и возвращает ноту для определения.
    static std::string GenerateIdentifyNote(bool isJustBeginned);
    //! Проверяет присланный ответ со сгенерированной нотой.
    static bool IsCorrectAnswer(const data::Degree note);
    //! Возвращает количество верных ответов.
    static std::uint8_t GetCorrectAnswers();
    //! Возвращает количество генераций.
    static std::uint8_t GetNotesGenerated();

    //
    //  Private methods.
    //
private:
    //! Обнуляет угаданное кол-во нот и текущее кол-во
сгенерированных нот.
    static void NullifyData();
    static bool IsDegreeEnumIndex(const std::uint8_t index);

    //
    //  Private members.
    //
private:
    //! Количество верно угаданных нот.
    inline static std::uint8_t correctAnswers_;
    //! Текущая сгенерированная нота.
    inline static data::Degree generatedNote_;
    //! Текущее количество сгенерированных нот.
    inline static std::uint8_t notesGenerated_;
};

};   // namespace elegia
```

backend/elegia/ScaleBuilder.h

```
#pragma once
#include <Types.h>

namespace elegia
{

//
//  Класс для работы с построителем тональности.
//

class ScaleBuilder
{
    //
    //  Public interface.
    //
public:
    //! Генерирует случайную тональность.
    static data::Scale GenerateTonality(const bool isJustBeginned);
    //! Строит и возвращает последовательность нот
тональности из заданных параметров
    static data::Scale BuildTonality(const data::ScaleRequestData&
scaleRequestData);
    //! Проверяет предложенную тональность пользователем.
    static bool IsCorrectAnswer(const data::ScaleRequestData&
scaleRequestData);
    //! Возвращает количество верных ответов.
    static std::uint8_t GetCorrectAnswers();
    //! Возвращает количество генераций.
    static std::uint8_t GetScalesGenerated();
    //! Возвращает параметры структуры - Degree,
AlterationSign, Key, по нотам тональности.
    static data::ScaleRequestData GetScaleRequestData(const
data::NoteVec& noteVec);

    //
    //  Private methods.
    //
private:
    //! Возвращает ноты тональности по параметрам структуры
- Degree, AlterationSign, Key.
    static data::NoteVec GetScale(const data::ScaleRequestData&
scaleRequestData);
    //! Возвращает ноты энгармонической тональности по
параметрам структуры - Degree,
```

```
    //! AlterationSign, Key.
    static data::NoteVecOpt GetEnharmonicScale(const
data::ScaleRequestData& scaleRequestData);
    //! Проверка успешности генерации тональности.
    static bool IsSuccessfulGeneration(
        const data::Scale& generatedScale, const bool isSuccessful,
const bool isJustBeginned);
    //! Обнуляет данные по угадыванию.
    static void NullifyData();
    //! Валидирует запрос.
    static bool ValidateScaleRequestData(const
data::ScaleRequestData& scaleRequestData);

    //
    //  Private members.
    //
private:
    //! Количество верно угаданных тональностей.
    inline static std::uint8_t correctAnswers_;
    //! Текущая сгенерированная тональность.
    inline static data::Scale generatedScale_;
    //! Текущее количество сгенерированных тональностей.
    inline static std::uint8_t scalesGenerated_;
};

}   // namespace elegia
```

backend/elegia/NotesIdentifier.cpp

```
#include "pch.h"

#include "NotesIdentifier.h"

#include <Note.h>

#include <Random.h>

namespace elegia
{

std::string NotesIdentifier::GenerateIdentifyNote(bool
isJustBeginned)
{
    // Срабатывает единоразово - при начале использования
тренажера определения нот.
    // Каждая попытка начать заново также должна обнулять
попытки.
    if (isJustBeginned)
    {
        NullifyData();
    }

    // Генерация индекса ступени и ее альтерации (диез, бемоль
или без)
    std::int8_t generatedNoteIndex;
    do
    {
        generatedNoteIndex = utils::Random::RandomizeInt32(
            data::Degree2Int8(data::Degree::C),
data::Degree2Int8(data::Degree::B));
        // Избегаем повторяемых нот...
    } while (!IsDegreeEnumIndex(generatedNoteIndex)
            || generatedNote_ == data::Degree{ generatedNoteIndex
});

    generatedNote_ = data::Degree{ generatedNoteIndex };
    data::Degree2String(generatedNote_);

    notesGenerated_++;

    // Формирование строкового результата
    return data::Degree2String(generatedNote_);
}

bool NotesIdentifier::IsCorrectAnswer(const data::Degree note)
{
    const bool isCorrect{ note == generatedNote_ };
    if (isCorrect)
    {
        correctAnswers_++;
    }

    return isCorrect;
}

std::uint8_t NotesIdentifier::GetCorrectAnswers()
{
    return correctAnswers_;
}

std::uint8_t NotesIdentifier::GetNotesGenerated()
{
    return notesGenerated_;
}

void NotesIdentifier::NullifyData()
{
    notesGenerated_ = 0;
    correctAnswers_ = 0;
}

bool NotesIdentifier::IsDegreeEnumIndex(const std::uint8_t index)
{
    switch (index)
    {
    case 1:
    case 3:
    case 5:
    case 6:
    case 8:
    case 10:
    case 12:
        return true;
    default:
        return false;
    }
}

}   // namespace elegia
```

backend/elegia/pch.h

```
#pragma once

//
//  STL
//

#include <map>
#include <optional>
#include <vector>
#include <utility>

//
//  FMT
//
#define FMT_HEADER_ONLY
#include <fmt/format.h>

//
//  App
//

#include <Types.h>
```

backend/elegia/ScaleBuilder.cpp

```
#include "pch.h"

#include "Note.h"
#include "Scale.h"

#include "ScaleBuilder.h"

#include <Random.h>
```

```
namespace scales
{

namespace major
{

static const data::NoteVec C{
    { data::Degree::C, data::AlterationSign::NONE }, // C
    { data::Degree::D, data::AlterationSign::NONE }, // D
    { data::Degree::E, data::AlterationSign::NONE }, // E
    { data::Degree::F, data::AlterationSign::NONE }, // F
    { data::Degree::G, data::AlterationSign::NONE }, // G
    { data::Degree::A, data::AlterationSign::NONE }, // A
    { data::Degree::B, data::AlterationSign::NONE }      // B
};

// Энгармонически равна Db-мажору
static const data::NoteVec CSharp{
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::SHARP },    // E#
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::B, data::AlterationSign::SHARP } // B#
};

// Энгармонически равна B-мажору
static const data::NoteVec CFlat{
    { data::Degree::C, data::AlterationSign::FLAT },  // Cb
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::FLAT },  // Fb
    { data::Degree::G, data::AlterationSign::FLAT },  // Gb
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT }       // Bb
};

static const data::NoteVec D{
    { data::Degree::D, data::AlterationSign::NONE },  // D
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },     // F#
    { data::Degree::G, data::AlterationSign::NONE },  // G
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP } // C#
};

// Энгармонически равна Eb-мажору
static const data::NoteVec DSharp{
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::F, data::AlterationSign::NONE },  // F
    { data::Degree::G, data::AlterationSign::NONE },  // G
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::C, data::AlterationSign::NONE },  // C
    { data::Degree::D, data::AlterationSign::NONE }       // D
};

// Энгармонически равна C#-мажору
static const data::NoteVec DFlat{
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },  // F
    { data::Degree::G, data::AlterationSign::FLAT },  // Gb
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE }      // C
};

// Энгармонически равна Fb-мажору
static const data::NoteVec E{
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },     // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP }     // D#
};

static const data::NoteVec EFlat{
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE }, // F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE }, // C
    { data::Degree::D, data::AlterationSign::NONE }      // D
};

static const data::NoteVec F{
    { data::Degree::F, data::AlterationSign::NONE }, // F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::NONE },// A
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE }, // C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::NONE }      // E
};

static const data::NoteVec FSharp{
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::SHARP } // E#
};

// Энгармонически равна Gb-мажору
static const data::NoteVec FFlat{
    { data::Degree::F, data::AlterationSign::FLAT },        // Fb
    { data::Degree::G, data::AlterationSign::FLAT },        // Gb
    { data::Degree::A, data::AlterationSign::FLAT },        // Ab
    { data::Degree::B, data::AlterationSign::DOUBLE_FLAT },   //
Bbb
    { data::Degree::C, data::AlterationSign::FLAT },        // Cb
    { data::Degree::D, data::AlterationSign::FLAT },        // Db
    { data::Degree::E, data::AlterationSign::FLAT }             // Eb
};

static const data::NoteVec G{
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::NONE },// A
    { data::Degree::B, data::AlterationSign::NONE }, // B
    { data::Degree::C, data::AlterationSign::NONE }, // C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::NONE }, // E
    { data::Degree::F, data::AlterationSign::SHARP } // F#
};

// Энгармонически равна Ab-мажору
static const data::NoteVec GSharp{
    { data::Degree::G, data::AlterationSign::SHARP },        // G#
    { data::Degree::A, data::AlterationSign::SHARP },        // A#
    { data::Degree::B, data::AlterationSign::SHARP },        // B#
    { data::Degree::C, data::AlterationSign::SHARP },        // C#
    { data::Degree::D, data::AlterationSign::SHARP },        // D#
    { data::Degree::E, data::AlterationSign::SHARP },        // E#
    { data::Degree::F, data::AlterationSign::DOUBLE_SHARP }
// F□
};

// Энгармонически равна F#-мажору
static const data::NoteVec GFlat{
    { data::Degree::G, data::AlterationSign::FLAT },  // Gb
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::FLAT },  // Cb
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE }      // F
};
```

```
static const data::NoteVec A{
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::NONE },  // D
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP }     // G#
};

// Энгармонически равна G#-мажору
static const data::NoteVec AFlat{
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE }      // G
};

// Энгармонически равна Cb-мажору
static const data::NoteVec B{
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP }     // A#
};

static const data::NoteVec BFlat{
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::NONE }      // A
};

}   // namespace major

namespace minor
{

static const data::NoteVec C{
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT }       // Bb
};

static const data::NoteVec CSharp{
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE }       // B
};

static const data::NoteVec D{
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::NONE },// E
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::NONE },// A
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE }      // C
};

// Энгармонически равна Eb-минору
static const data::NoteVec DSharp{
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::SHARP },    // E#
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP } // C#
};

// Энгармонически равна C#-минору
static const data::NoteVec DFlat{
    { data::Degree::D, data::AlterationSign::FLAT },        // Db
    { data::Degree::E, data::AlterationSign::FLAT },        // Eb
    { data::Degree::F, data::AlterationSign::FLAT },        // Fb
    { data::Degree::G, data::AlterationSign::FLAT },        // Gb
    { data::Degree::A, data::AlterationSign::FLAT },        // Ab
    { data::Degree::B, data::AlterationSign::DOUBLE_FLAT },   //
Bbb
    { data::Degree::C, data::AlterationSign::FLAT }             // Cb
};

static const data::NoteVec E{
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::NONE },  // G
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::NONE },  // C
    { data::Degree::D, data::AlterationSign::NONE }      // D
};

// Энгармонически равна F-минору
static const data::NoteVec ESharp{
    { data::Degree::E, data::AlterationSign::SHARP },          // E#
    { data::Degree::F, data::AlterationSign::DOUBLE_SHARP },
   // F□
    { data::Degree::G, data::AlterationSign::SHARP },          // G#
    { data::Degree::A, data::AlterationSign::SHARP },          // A#
    { data::Degree::B, data::AlterationSign::SHARP },          // B#
    { data::Degree::C, data::AlterationSign::SHARP },          // C#
    { data::Degree::D, data::AlterationSign::SHARP }           // D#
};

// Энгармонически равна D#-минору
static const data::NoteVec EFlat{
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::FLAT },  // Gb
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::FLAT },  // Cb
    { data::Degree::D, data::AlterationSign::FLAT }       // Db
};

// Энгармонически равна E#-минору
static const data::NoteVec F{
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE },// G
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT }       // Eb
};

static const data::NoteVec FSharp{
    { data::Degree::F, data::AlterationSign::NONE },  // F#
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::NONE },  // A
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::NONE },  // D
    { data::Degree::E, data::AlterationSign::NONE }       // E
};

static const data::NoteVec G{
    { data::Degree::G, data::AlterationSign::NONE },// G
```

```
    { data::Degree::A, data::AlterationSign::NONE },// A
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE }      // F
};

// Энгармонически равна Ab-минору
static const data::NoteVec GSharp{
    { data::Degree::G, data::AlterationSign::SHARP },    // G#
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP }  // F#
};

static const data::NoteVec A{
    { data::Degree::A, data::AlterationSign::NONE },// A
    { data::Degree::B, data::AlterationSign::NONE },// B
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::NONE },// D
    { data::Degree::E, data::AlterationSign::NONE },// E
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::NONE }      // G
};

static const data::NoteVec ASharp{
    { data::Degree::A, data::AlterationSign::SHARP },    // A#
    { data::Degree::B, data::AlterationSign::SHARP },    // B#
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::SHARP },    // D#
    { data::Degree::E, data::AlterationSign::SHARP },    // E#
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::SHARP }     // G#
};

// Энгармонически равна G#-минору
static const data::NoteVec AFlat{
    { data::Degree::A, data::AlterationSign::FLAT },  // Ab
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::FLAT },  // Cb
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::FLAT },  // Fb
    { data::Degree::G, data::AlterationSign::FLAT }      // Gb
};

static const data::NoteVec B{
    { data::Degree::B, data::AlterationSign::NONE },  // B
    { data::Degree::C, data::AlterationSign::SHARP },    // C#
    { data::Degree::D, data::AlterationSign::NONE },  // D
    { data::Degree::E, data::AlterationSign::NONE },  // E
    { data::Degree::F, data::AlterationSign::SHARP },    // F#
    { data::Degree::G, data::AlterationSign::NONE },  // G
    { data::Degree::A, data::AlterationSign::NONE }      // A
};

// Энгармонически равна A#-минору
static const data::NoteVec BFlat{
    { data::Degree::B, data::AlterationSign::FLAT },  // Bb
    { data::Degree::C, data::AlterationSign::NONE },// C
    { data::Degree::D, data::AlterationSign::FLAT },  // Db
    { data::Degree::E, data::AlterationSign::FLAT },  // Eb
    { data::Degree::F, data::AlterationSign::NONE },// F
    { data::Degree::G, data::AlterationSign::FLAT },  // Gb
    { data::Degree::A, data::AlterationSign::FLAT }      // Ab
};

}   // namespace minor

}   // namespace scales

namespace details
{

static const std::multimap<data::ScaleRequestData,
data::NoteVec> ScaleMap{
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::C },
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::CSharp },
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::CFlat },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::D },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::DSharp },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::DFlat },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::E },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::EFlat },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::F },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::FSharp },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::FFlat },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::G },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::GSharp },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::GFlat },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::A },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::AFlat },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::B },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::BFlat },
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::C },
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::CSharp },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::D },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::DSharp },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::DFlat },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::E },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::ESharp },
```

```
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::EFlat },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::F },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::FSharp },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::G },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::GSharp },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::A },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::AFlat },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::B },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::BFlat }
};

static const std::multimap<data::NoteVec,
data::ScaleRequestData> ScalesHeadersMap{
    {
      scales::major::C,
      data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::CSharp,
      data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
      scales::major::CFlat,
      data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::D,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::DSharp,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
      scales::major::DFlat,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::E,
      data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::EFlat,
      data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::F,
      data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::FSharp,
      data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
      scales::major::FFlat,
      data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::G,
      data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::GSharp,
      data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
      scales::major::GFlat,
      data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::A,
      data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::AFlat,
      data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::major::B,
      data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
      scales::major::BFlat,
      data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
      scales::minor::C,
      data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
      scales::minor::CSharp,
      data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
      scales::minor::D,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
      scales::minor::DSharp,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
      scales::minor::DFlat,
      data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    {
      scales::minor::E,
      data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
      scales::minor::ESharp,
      data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
```

```
    {
        scales::minor::EFlat,
        data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    {
        scales::minor::F,
        data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
        scales::minor::FSharp,
        data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::G,
        data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
        scales::minor::GSharp,
        data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::A,
        data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
        scales::minor::AFlat,
        data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    {
        scales::minor::B,
        data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
        scales::minor::BFlat,
        data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    // Энгармонические
    {
        scales::major::DFlat,
        data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
        scales::major::B,
        data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
        scales::major::EFlat,
        data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
        scales::major::CSharp,
        data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
        scales::major::FFlat,
        data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
        scales::major::GFlat,
        data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
        scales::major::AFlat,
        data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MAJOR },
    },
    {
        scales::major::FSharp,
        data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
        scales::major::GSharp,
        data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MAJOR },
    },
    {
        scales::major::CFlat,
        data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MAJOR },
    },
    {
        scales::minor::EFlat,
        data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::CSharp,
        data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    {
        scales::minor::F,
        data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::DSharp,
        data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MINOR },
    },
    {
        scales::minor::ESharp,
        data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MINOR },
    },
    {
        scales::minor::AFlat,
        data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::GSharp,
        data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::SHARP, data::Key::MINOR },
    },
    {
        scales::minor::ASharp,
        data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MINOR },
    }
};

static const std::multimap<data::ScaleRequestData,
data::NoteVec> EnharmonicScaleMap{
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::DFlat },
    { data::ScaleRequestData{ data::Degree::C,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::B },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::EFlat },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::CSharp },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::FFlat },
```

```
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::GFlat },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MAJOR },
      scales::major::AFlat },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::FSharp },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::FLAT, data::Key::MAJOR },
      scales::major::GSharp },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::NONE, data::Key::MAJOR },
      scales::major::CFlat },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::EFlat },
    { data::ScaleRequestData{ data::Degree::D,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::CSharp },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::F },
    { data::ScaleRequestData{ data::Degree::E,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::DSharp },
    { data::ScaleRequestData{ data::Degree::F,
data::AlterationSign::NONE, data::Key::MINOR },
      scales::minor::ESharp },
    { data::ScaleRequestData{ data::Degree::G,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::AFlat },
    { data::ScaleRequestData{ data::Degree::A,
data::AlterationSign::SHARP, data::Key::MINOR },
      scales::minor::GSharp },
    { data::ScaleRequestData{ data::Degree::B,
data::AlterationSign::FLAT, data::Key::MINOR },
      scales::minor::ASharp }
};

}   // namespace details

namespace elegia
{

data::Scale ScaleBuilder::GenerateTonality(const bool
isJustBeginned)
{
    // Срабатывает единоразово - при начале использования
тренажера построителя тональностей.
    // Каждая попытка начать заново также должна обнулять
попытки.
    if (isJustBeginned)
    {
        NullifyData();
    }

    bool tonalityExists = false;
    data::ScaleRequestData scaleRequestData;
    data::Scale generatedScale;
    std::int8_t generatedSignIndex, generatedKeyIndex;
    do
    {
        static const std::vector<data::Degree> degreesVec{
            data::Degree::C, data::Degree::D, data::Degree::E,
data::Degree::F,
            data::Degree::G, data::Degree::A, data::Degree::B,
        };

        generatedSignIndex = utils::Random::RandomizeInt32(0, 2);
        generatedKeyIndex = utils::Random::RandomizeInt32(0, 1);

        scaleRequestData.tonic_ =
            degreesVec[utils::Random::RandomizeInt32(0,
degreesVec.size() - 1)];
        scaleRequestData.alterationSign_ = data::AlterationSign{
generatedSignIndex };
        scaleRequestData.key_ = data::Key{ generatedKeyIndex };

        try
        {
            generatedScale = BuildTonality(scaleRequestData);
            tonalityExists = true;
        }
        catch (...)
        {
            tonalityExists = false;
        }

        //! TODO: возможно наличие здесь лямбда функции было
бы правильней.
    } while (!IsSuccessfulGeneration(generatedScale, tonalityExists,
isJustBeginned));

    generatedScale_ = generatedScale;
    scalesGenerated_++;

    return generatedScale_;
}

data::Scale ScaleBuilder::BuildTonality(const
data::ScaleRequestData& scaleRequestData)
{
    const auto scaleNotes{ GetScale(scaleRequestData) };

    data::Scale scale;
    scale.tonic_.degree_ = scaleRequestData.tonic_;
    scale.tonic_.alterationSign_ = scaleRequestData.alterationSign_;
    scale.key_ = scaleRequestData.key_;
    scale.notes_ = scaleNotes;
    scale.enharmonicScale_ =
GetEnharmonicScale(scaleRequestData);

    return scale;
}

bool ScaleBuilder::IsCorrectAnswer(const
data::ScaleRequestData& scaleRequestData)
{
    const bool isMainScale{ scaleRequestData.tonic_ ==
generatedScale_.tonic_.degree_
                            && scaleRequestData.alterationSign_
                               == generatedScale_.tonic_.alterationSign_
                            && scaleRequestData.key_ ==
generatedScale_.key_ };
    data::BoolOpt isEnharmonic;
    if (generatedScale_.enharmonicScale_.has_value())
    {
        try
        {
            isEnharmonic =
                BuildTonality(scaleRequestData).notes_ ==
generatedScale_.enharmonicScale_.value();
        }
        catch (...)
        {
            isEnharmonic = std::nullopt;
        }
    }

    const bool isCorrect{ isMainScale || (isEnharmonic.has_value()
&& isEnharmonic.value()) };
    if (isCorrect)
    {
        correctAnswers_++;
    }

    return isCorrect;
}

std::uint8_t ScaleBuilder::GetCorrectAnswers()
{
    return correctAnswers_;
}
```

```
std::uint8_t ScaleBuilder::GetScalesGenerated()
{
    return scalesGenerated_;
}

data::ScaleRequestData ScaleBuilder::GetScaleRequestData(const
data::NoteVec& noteVec)
{
    for (auto it = details::ScalesHeadersMap.begin(); it !=
details::ScalesHeadersMap.end(); ++it)
    {
        if (noteVec == it->first)
        {
            return it->second;
        }
    }

    throw std::runtime_error("Not yet implemented tonality");
}

data::NoteVec ScaleBuilder::GetScale(const
data::ScaleRequestData& scaleRequestData)
{
    for (auto it = details::ScaleMap.begin(); it !=
details::ScaleMap.end(); ++it)
    {
        if (scaleRequestData == it->first)
        {
            return it->second;
        }
    }

    throw std::runtime_error("Not yet implemented tonality");
}

data::NoteVecOpt ScaleBuilder::GetEnharmonicScale(const
data::ScaleRequestData& scaleRequestData)
{
    const auto result{
details::EnharmonicScaleMap.find(scaleRequestData) };
    if (result == details::EnharmonicScaleMap.end())
    {
        return std::nullopt;
    }

    return result->second;
}

bool ScaleBuilder::IsSuccessfulGeneration(
    const data::Scale& generatedScale, const bool tonalityExists,
const bool isJustBeginned)
{
    // При первой проверке - предыдущих тональностей не
может быть.
    if (isJustBeginned)
    {
        return tonalityExists;
    }
    // При следующих проверках - проверка с предыдущей
тональностью и на энгармоническую.
    else
    {
        return (tonalityExists && generatedScale_ !=
generatedScale);
    }
}

void ScaleBuilder::NullifyData()
{
    correctAnswers_ = 0;
    generatedScale_ = {};
    scalesGenerated_ = 0;
}

bool ScaleBuilder::ValidateScaleRequestData(const
data::ScaleRequestData& scaleRequestData)
{
    try
    {
        GetScale(scaleRequestData);

        return true;
    }
    catch (...)
    {
        return false;
    }
}

}   // namespace elegia
```

backend/test/ElegiaNotesIdentifierTest.cpp

```
#include "pch.h"

#include <../data/include/Enums.h>

#include <../elegia/include/NotesIdentifier.h>

#include <../utils/include/Random.h>

/**
 * Описание: обработка корректных нот
 * Когда: сгенерированные ноты совпадают с предложенными
пользователем ответами
 * Тогда: функция возвращает true
 **/
TEST(ElegiaNotesIdentifierTest, CorrectNote)
{
    for (int i = 0; i < 3; i++)
    {
        const auto generatedNote{
elegia::NotesIdentifier::GenerateIdentifyNote(true) };
        const auto checkResult{
elegia::NotesIdentifier::IsCorrectAnswer(
            data::String2Degree(generatedNote)) };

        ASSERT_TRUE(checkResult);
    }

    ASSERT_EQ(
        elegia::NotesIdentifier::GetCorrectAnswers(),
elegia::NotesIdentifier::GetNotesGenerated());
}

/**
 * Описание: обработка некорректной ноты
 * Когда: сгенерированная нота не совпадает с предложенным
пользователем ответом
 * Тогда: функция возвращает false
 **/
TEST(ElegiaNotesIdentifierTest, WrongNote)
{
    const data::Degree generatedNote{ data::String2Degree(
        elegia::NotesIdentifier::GenerateIdentifyNote(true)) };

    data::Degree wrongNote;
    while (wrongNote == generatedNote)
    {
        wrongNote = data::Degree{ utils::Random::RandomizeInt32(
            data::Degree2Int8(data::Degree::C),
data::Degree2Int8(data::Degree::B)) };
    }

    const auto checkResult{
elegia::NotesIdentifier::IsCorrectAnswer(wrongNote) };

    ASSERT_FALSE(checkResult);
    ASSERT_NE(elegia::NotesIdentifier::GetCorrectAnswers(),
elegia::NotesIdentifier::GetNotesGenerated());
}

/**
```

```
 * Описание: проверка гарантии неповторяемости ноты подряд
 * Когда: из 100 сгенерированных нот...
 * Тогда: ...каждые две подряд не будут повторятся
 **/
TEST(ElegiaNotesIdentifierTest, NoRepeatableNotesInARow)
{
    data::Degree generatedNote1, generatedNote2;
    generatedNote1 =
data::String2Degree(elegia::NotesIdentifier::GenerateIdentifyNote(
true));
    for (int i = 0; i < 100; i++)
    {
        generatedNote2 =
data::String2Degree(elegia::NotesIdentifier::GenerateIdentifyNote(
false));
        ASSERT_NE(generatedNote1, generatedNote2);

        generatedNote1 =
data::String2Degree(elegia::NotesIdentifier::GenerateIdentifyNote(
false));
    }
}
```

backend/test/ElegiaScaleBuilder Test.cpp

```
#include "pch.h"

#include <../data/include/Types.h>
#include <../data/include/Note.h>
#include <../data/include/Scale.h>

#include <../elegia/include/ScaleBuilder.h>

/**
 * Описание: постройка тональности и ее угадывание
 * Когда: при постройке тональности
 * Тогда: построенная тональность соответствует
действительной тональности
 **/
TEST(ElegiaScaleBuilderTest, BuildAndCheckScale)
{
    const data::ScaleRequestData scaleRequestData{
        data::Degree::C,
        data::AlterationSign::NONE,
        data::Key::MINOR,
    };

    const auto builtScale{
elegia::ScaleBuilder::BuildTonality(scaleRequestData) };

    ASSERT_EQ(scaleRequestData.tonic_,
builtScale.tonic_.degree_);
    ASSERT_EQ(scaleRequestData.alterationSign_,
builtScale.tonic_.alterationSign_);
    ASSERT_EQ(scaleRequestData.key_, builtScale.key_);

    data::NoteVec expectedScale;
    expectedScale.push_back(data::Note{ data::Degree::C,
data::AlterationSign::NONE });
    expectedScale.push_back(data::Note{ data::Degree::D,
data::AlterationSign::NONE });
    expectedScale.push_back(data::Note{ data::Degree::E,
data::AlterationSign::FLAT });
    expectedScale.push_back(data::Note{ data::Degree::F,
data::AlterationSign::NONE });
    expectedScale.push_back(data::Note{ data::Degree::G,
data::AlterationSign::NONE });
    expectedScale.push_back(data::Note{ data::Degree::A,
data::AlterationSign::FLAT });
    expectedScale.push_back(data::Note{ data::Degree::B,
data::AlterationSign::FLAT });

    ASSERT_EQ(builtScale.notes_[0], expectedScale[0]);
    ASSERT_EQ(builtScale.notes_[1], expectedScale[1]);
    ASSERT_EQ(builtScale.notes_[2], expectedScale[2]);
    ASSERT_EQ(builtScale.notes_[3], expectedScale[3]);
    ASSERT_EQ(builtScale.notes_[4], expectedScale[4]);
    ASSERT_EQ(builtScale.notes_[5], expectedScale[5]);
    ASSERT_EQ(builtScale.notes_[6], expectedScale[6]);
}

/**
 * Описание: генерация тональности и ее угадывание с верным
ответом
 * Когда: сгенерированная тональность совпадает с
предложенным пользователем ответом
 * Тогда: функция проверки возвращает true
 **/
TEST(ElegiaScaleBuilderTest, CorrectTonality)
{
    const auto generatedTonality{
elegia::ScaleBuilder::GenerateTonality(true) };

    data::ScaleRequestData scaleRequestData;
    scaleRequestData.tonic_ = generatedTonality.tonic_.degree_;
    scaleRequestData.alterationSign_ =
generatedTonality.tonic_.alterationSign_;
    scaleRequestData.key_ = generatedTonality.key_;

    ASSERT_TRUE(elegia::ScaleBuilder::IsCorrectAnswer(scaleR
equestData));
}

/**
 * Описание: генерация тональности и ее угадывание с верным
ответом
 * Когда: сгенерированная тональность совпадает с
предложенным пользователем ответом
 * Тогда: функция проверки возвращает false
 **/
TEST(ElegiaScaleBuilderTest, WrongTonality)
{
    const auto generatedTonality{
elegia::ScaleBuilder::GenerateTonality(true) };

    // Вводим рандомную дату для неверного ответа
    data::ScaleRequestData scaleRequestData;
    scaleRequestData.tonic_ = data::Degree::B;
    scaleRequestData.alterationSign_ =
data::AlterationSign::BEKAR;
    scaleRequestData.key_ = data::Key::MAJOR;

    ASSERT_FALSE(elegia::ScaleBuilder::IsCorrectAnswer(scale
RequestData));
}

/**
 * Описание: ввод энгармонической тональности после
генерации
 * Когда: вводится энгармоническая тональность
 * Тогда: ответ так же считается верным
 **/
TEST(ElegiaScaleBuilderTest, EnharmonicTonality)
{
    const auto generatedTonality{
elegia::ScaleBuilder::GenerateTonality(true) };
    const auto enharmonicTonality{
elegia::ScaleBuilder::GetScaleRequestData(generatedTonality.note
s_) };

    data::ScaleRequestData scaleRequestData;
    scaleRequestData.tonic_ = enharmonicTonality.tonic_;
    scaleRequestData.alterationSign_ =
enharmonicTonality.alterationSign_;
    scaleRequestData.key_ = enharmonicTonality.key_;

    ASSERT_TRUE(elegia::ScaleBuilder::IsCorrectAnswer(scaleR
equestData));
}

/**
 * Описание: проверка гарантии неповторяемости тональности
подряд
```

```
 * Когда: из 100 сгенерированных тональностей...
 * Тогда: ...каждые две подряд не будут повторятся
 **/
TEST(ElegiaScaleBuilderTest, NoRepeatableScalesInARow)
{
    data::Scale generatedScale1, generatedScale2;
    generatedScale1 = elegia::ScaleBuilder::GenerateTonality(true);
    for (int i = 0; i < 100; i++)
    {
        generatedScale2 =
elegia::ScaleBuilder::GenerateTonality(false);
        ASSERT_NE(generatedScale2, generatedScale1);

        generatedScale1 =
elegia::ScaleBuilder::GenerateTonality(false);
    }
}
```

backend/test/EnumConversion Test.cpp

```
#include "pch.h"

#include <../data/include/Enums.h>
#include <../data/include/Note.h>

/**
 * Описание: выполнение конвертаций из Degree enum в
std::string и наоборот
 * Когда: выполняется коневертация из enum/std::string
 * Тогда: успешно возвращается конвертированный
std::string/enum
 **/
TEST(EnumConversionTest, DegreeConversion)
{
    // clang-format off
    // To enum (string)
    ASSERT_EQ(data::Degree::C,      data::String2Degree("C"));
    ASSERT_EQ(data::Degree::D,      data::String2Degree("D"));
    ASSERT_EQ(data::Degree::E,      data::String2Degree("E"));
    ASSERT_EQ(data::Degree::F,      data::String2Degree("F"));
    ASSERT_EQ(data::Degree::G,      data::String2Degree("G"));
    ASSERT_EQ(data::Degree::A,      data::String2Degree("A"));
    ASSERT_EQ(data::Degree::B,      data::String2Degree("B"));

    // To string
    ASSERT_EQ("C",     data::Degree2String(data::Degree::C));
    ASSERT_EQ("D",     data::Degree2String(data::Degree::D));
    ASSERT_EQ("E",     data::Degree2String(data::Degree::E));
    ASSERT_EQ("F",     data::Degree2String(data::Degree::F));
    ASSERT_EQ("G",     data::Degree2String(data::Degree::G));
    ASSERT_EQ("A",     data::Degree2String(data::Degree::A));
    ASSERT_EQ("B",     data::Degree2String(data::Degree::B));


    // To enum (int)
    ASSERT_EQ(data::Degree::C,      data::Degree{1});
    ASSERT_EQ(data::Degree::D,      data::Degree{3});
    ASSERT_EQ(data::Degree::E,      data::Degree{5});
    ASSERT_EQ(data::Degree::F,      data::Degree{6});
    ASSERT_EQ(data::Degree::G,      data::Degree{8});
    ASSERT_EQ(data::Degree::A,      data::Degree{10});
    ASSERT_EQ(data::Degree::B,      data::Degree{12});
    // clang-format on
}

/**
 * Описание: выполнение конвертаций из Octave enum в
std::string и наоборот
 * Когда: выполняется коневертация из enum/std::string
 * Тогда: успешно возвращается конвертированный
std::string/enum
 **/
TEST(EnumConversionTest, OctaveConversion)
{
    // clang-format off
    // To enum
    ASSERT_EQ(data::Octave::SUB_CONTRA,
data::String2Octave("0"));
    ASSERT_EQ(data::Octave::CONTRA,
data::String2Octave("1"));
    ASSERT_EQ(data::Octave::GREAT,
data::String2Octave("2"));
    ASSERT_EQ(data::Octave::SMALL,
data::String2Octave("3"));
    ASSERT_EQ(data::Octave::LINE1,
data::String2Octave("4"));
    ASSERT_EQ(data::Octave::LINE2,
data::String2Octave("5"));
    ASSERT_EQ(data::Octave::LINE3,
data::String2Octave("6"));
    ASSERT_EQ(data::Octave::LINE4,
data::String2Octave("7"));
    ASSERT_EQ(data::Octave::LINE5,
data::String2Octave("8"));
    ASSERT_EQ(data::Octave::LINE6,
data::String2Octave("9"));

    // To string
    ASSERT_EQ("0",
data::Octave2String(data::Octave::SUB_CONTRA));
    ASSERT_EQ("1",
data::Octave2String(data::Octave::CONTRA));
    ASSERT_EQ("2",
data::Octave2String(data::Octave::GREAT));
    ASSERT_EQ("3",
data::Octave2String(data::Octave::SMALL));
    ASSERT_EQ("4",
data::Octave2String(data::Octave::LINE1));
    ASSERT_EQ("5",
data::Octave2String(data::Octave::LINE2));
    ASSERT_EQ("6",
data::Octave2String(data::Octave::LINE3));
    ASSERT_EQ("7",
data::Octave2String(data::Octave::LINE4));
    ASSERT_EQ("8",
data::Octave2String(data::Octave::LINE5));
    ASSERT_EQ("9",
data::Octave2String(data::Octave::LINE6));
    // clang-format on
}

/**
 * Описание: выполнение конвертаций из AlterationSign enum
в std::string и наоборот
 * Когда: выполняется коневертация из enum/std::string
 * Тогда: успешно возвращается конвертированный
std::string/enum
 **/
TEST(EnumConversionTest, AlterationSignConversion)
{
    // clang-format off
    // To enum
    ASSERT_EQ(data::AlterationSign::NONE,
data::String2AlterationSign(""));
    ASSERT_EQ(data::AlterationSign::SHARP,
data::String2AlterationSign("#"));
    ASSERT_EQ(data::AlterationSign::FLAT,
data::String2AlterationSign("b"));
    ASSERT_EQ(data::AlterationSign::DOUBLE_SHARP,
data::String2AlterationSign("##"));
    ASSERT_EQ(data::AlterationSign::DOUBLE_FLAT,
data::String2AlterationSign("bb"));
    ASSERT_EQ(data::AlterationSign::BEKAR,
data::String2AlterationSign("♮"));

    // To string
    ASSERT_EQ("",
data::AlterationSign2String(data::AlterationSign::NONE));
    ASSERT_EQ("#",
data::AlterationSign2String(data::AlterationSign::SHARP));
    ASSERT_EQ("b",
data::AlterationSign2String(data::AlterationSign::FLAT));
```

```
    ASSERT_EQ("##",
data::AlterationSign2String(data::AlterationSign::DOUBLE_SHA
RP));
    ASSERT_EQ("bb",
data::AlterationSign2String(data::AlterationSign::DOUBLE_FLA
T));
    ASSERT_EQ("♮",
data::AlterationSign2String(data::AlterationSign::BEKAR));
    // clang-format on
}

/**
 * Описание: выполнение конвертаций из Note в std::string
 * Когда: выполняется коневертация из Note
 * Тогда: успешно возвращается конвертированный std::string
 **/
TEST(EnumConversionTest, NoteConversion)
{
    {
        data::Note note;
        note.degree_ = data::Degree::C;
        note.alterationSign_ = data::AlterationSign::FLAT;
        note.octave_ = data::Octave::SMALL;

        // To string
        ASSERT_EQ(data::Note2String(note, true), "Cb3");
        ASSERT_EQ(data::Note2String(note, false), "Cb");
    }

    // No octave/alteration
    {
        data::Note note;
        note.degree_ = data::Degree::E;

        // To string
        ASSERT_EQ(data::Note2String(note, false), "E");
    }
}

/**
 * Описание: выполнение конвертаций из Key enum в
std::string и наоборот
 * Когда: выполняется коневертация из enum/std::string
 * Тогда: успешно возвращается конвертированный
std::string/enum
 **/
TEST(EnumConversionTest, KeyConversion)
{
    // clang-format off
    // To enum
    ASSERT_EQ(data::Key::MAJOR,
data::String2Key("Major"));
    ASSERT_EQ(data::Key::MINOR,
data::String2Key("Minor"));

    // To string
    ASSERT_EQ("Major",
data::Key2String(data::Key::MAJOR));
    ASSERT_EQ("Minor",
data::Key2String(data::Key::MINOR));
    // clang-format on
}
```

backend/test/Main

```
#include "pch.h"

// TODO: uncomment once database appears (it is planned)
//       and also dont forget to create this file and add into global test
env
// #include "Environment.h"


int main(int argc, char* argv[])
{
    testing::InitGoogleTest(&argc, argv);
    // If you need to run only one unit test - uncomment this code
and put your unit test name
    // testing::GTEST_FLAG(filter) = "UnitTestName*";

    // If you need to run only one case of unit test - uncomment this
code and put your case name
    // testing::GTEST_FLAG(filter) = "UnitTestName.CaseName";

    // TODO: related to the 'Environment.h' to-do.
    // GTest is an owner of WorkFactory
    //testing::AddGlobalTestEnvironment(new
testing::Environment);

    // TODO: related to the 'Environment.h' to-do.
    //testing::TestEventListeners& listeners =
testing::UnitTest::GetInstance()->listeners();

    // TODO: related to the 'Environment.h' to-do.
    //listeners.Append(new test::WorkFactoryResetter);

    return RUN_ALL_TESTS();
}
```

backend/test/NoteTest.cpp

```
#include "pch.h"

#include <../data/include/Note.h>

/**
 * Описание: инкрементация структуры Note
 * Когда: вызвана инкрементация operator++
 * Тогда: нота инкрементирована корректно
 **/
TEST(NoteTest, Incremention)
{
    {
        data::Note note{ data::Degree::C,
data::AlterationSign::NONE, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::C,
                                 data::AlterationSign::SHARP,
                                 data::Octave::LINE1 };
        ++note;

        ASSERT_EQ(note, expectedNote);
    }

    {
        data::Note note{ data::Degree::D,
data::AlterationSign::FLAT, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::D,
data::AlterationSign::NONE, data::Octave::LINE1 };
        ++note;

        ASSERT_EQ(note, expectedNote);
    }

    {
        data::Note note{ data::Degree::E,
data::AlterationSign::SHARP, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::F,
data::AlterationSign::SHARP, data::Octave::LINE1 };
        ++note;

        ASSERT_EQ(note, expectedNote);
    }
}

/**
 * Описание: декрементация структуры Note
 * Когда: вызвана инкрементация operator--
 * Тогда: нота декрементирована корректно
 **/
TEST(NoteTest, Decremention)
{
    {
        data::Note note{ data::Degree::C,
data::AlterationSign::NONE, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::B,
data::AlterationSign::NONE, data::Octave::SMALL };
```

```
        --note;

        ASSERT_EQ(note, expectedNote);
    }

    {
        data::Note note{ data::Degree::D,
data::AlterationSign::FLAT, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::C,
data::AlterationSign::NONE, data::Octave::LINE1 };
        --note;

        ASSERT_EQ(note, expectedNote);
    }

    {
        data::Note note{ data::Degree::E,
data::AlterationSign::SHARP, data::Octave::LINE1 };
        data::Note expectedNote{ data::Degree::E,
data::AlterationSign::NONE, data::Octave::LINE1 };
        --note;

        ASSERT_EQ(note, expectedNote);
    }
}

/**
 * Описание: сравнение структур Note
 * Когда: вызваны операторы сравнения operator> и operator<
 * Тогда: ноты сравниваются правильно
 **/
TEST(NoteTest, Comparement)
{
    {
        data::Note note1{ data::Degree::E,
data::AlterationSign::SHARP, data::Octave::LINE1 };
        data::Note note2{ data::Degree::E,
data::AlterationSign::NONE, data::Octave::LINE1 };

        ASSERT_TRUE(note1 > note2);
        ASSERT_FALSE(note1 < note2);
    }

    {
        data::Note note1{ data::Degree::D,
data::AlterationSign::NONE, data::Octave::LINE1 };
        data::Note note2{ data::Degree::E,
data::AlterationSign::NONE, data::Octave::LINE1 };

        ASSERT_TRUE(note1 < note2);
        ASSERT_FALSE(note1 > note2);
    }

    {
        data::Note note1{ data::Degree::C,
data::AlterationSign::SHARP, data::Octave::LINE2 };
        data::Note note2{ data::Degree::C,
data::AlterationSign::NONE, data::Octave::LINE1 };

        ASSERT_TRUE(note1 > note2);
        ASSERT_FALSE(note1 < note2);
    }

    {
        data::Note note1{ data::Degree::F,
data::AlterationSign::SHARP, data::Octave::LINE1 };
        data::Note note2{ data::Degree::A,
data::AlterationSign::FLAT, data::Octave::SMALL };

        ASSERT_TRUE(note1 > note2);
        ASSERT_FALSE(note1 < note2);
    }
}
```

backend/test/pch.h

```
#pragma once

//
// STL
//
#include <iostream>
#include <string>

//
// GTest
//
#include <gmock/gmock.h>
#include <gtest/gtest.h>
```

backend/test/RandomTest.cpp

```
#include "pch.h"

#include <../utils/include/Random.h>

/**
 * Описание: рандомизация целого 32-битного числа
 * Когда: вызвана функция рандомизации
 * Тогда: результат находится в пределах, указанных в
параметрах (включительно верх-низ)
**/
TEST(RandomTest, RandomizeInt32)
{
    const auto lowBound = -100;
    const auto highBound = 100;

    // Из десяти испытаний результат должен входить в
диапазон рандомизации.
    for (int i = 0; i < 10; i++)
    {
        const auto randomizedInt{
utils::Random::RandomizeInt32(lowBound, highBound) };

        ASSERT_TRUE(randomizedInt >= lowBound &&
randomizedInt <= highBound);
        ASSERT_FALSE(randomizedInt < lowBound ||
randomizedInt > highBound);
    }
}
```

backend/test/Random.h

```
#pragma once

namespace utils
{

class Random
{
    //
    //  Public interface.
    //
public:
    //! Генерирует случайное число int32_t от lowBoundExcl
(включительно) до highBoundExcl (включительно)
    static std::int32_t RandomizeInt32(std::int32_t lowBoundExcl,
std::int32_t highBoundExcl);
};


}   // namespace utils
```

backend/test/pch.h

```
#pragma once

//
//  STL
//

#include <random>

//
//  FMT
```

```
//
#define FMT_HEADER_ONLY
#include <fmt/format.h>
```

backend/test/Random.cpp

```
#include "pch.h"

#include "Random.h"

namespace utils
{

std::int32_t Random::RandomizeInt32(std::int32_t lowBoundExcl,
std::int32_t highBoundExcl)
{
   if (highBoundExcl < lowBoundExcl)
   {
      throw(std::invalid_argument(fmt::format("Invalid boundaries:
lowBoundExcl = {}, highBoundExcl = {}")));
   }

   // Random generator.
   std::random_device rd;
   std::mt19937 gen(rd());
   std::uniform_int_distribution<> dist(lowBoundExcl,
highBoundExcl);

   return dist(gen);
}

}
```