ЗМІСТ

[ВСТУП 6](#_Toc470724779)

[1 РОЗГЛЯД СУЧАСНОГО ПЗ ДЛЯ ВІЗУАЛІЗАЦІЇ АНАЛІЗУ БІОМЕДИНИХ ДАНИХ. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ 8](#_Toc470724780)

[1.1 Аналіз технічного завдання. 8](#_Toc470724781)

[1.2 Розгляд сучасного програмного забезпечення для візуалізації аналізу біомединих даних 9](#_Toc470724782)

[2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 15](#_Toc470724783)

[2.1 Діаграма варіантів використання (use case diagram) 15](#_Toc470724784)

[2.2 Основні компоненти. Діаграма компонентів (component diagram) 17](#_Toc470724785)

[2.2.1. Головне вікно 19](#_Toc470724786)

[2.2.2. UI Engine та Templates 20](#_Toc470724787)

[2.2.3. Spectrum Engine 20](#_Toc470724788)

[3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 22](#_Toc470724789)

[3.1 Обґрунтування вибору мови програмування 22](#_Toc470724790)

[3.1.1 Вибір мови програмування 22](#_Toc470724791)

[3.1.2 Вибір бібліотеки класів 23](#_Toc470724792)

[3.2 Реалізація головного вікна 27](#_Toc470724793)

[3.3 Реалізація Spectrum Engine, класів обробки та відтворення вхідного потоку 30](#_Toc470724794)

[ВИСНОВКИ 39](#_Toc470724795)

[ЛІТЕРАТУРА 40](#_Toc470724796)

[ДОДАТКИ 41](#_Toc470724797)

# ВСТУП

В останні 20-30 років медицина і біологія вступили у нову фазу свого розвитку. Нагромадження величезних масивів кількісних даних і доступність

обчислювальної техніки посилили математизацію біології та медицини. Сьогодні вже неможливо собі уявити наукові дослідження в галузі біології та медицини, які виконані без статистичного аналізу отриманих фактичних даних. Біостатистика стала суттєвим кроком у русі медичного дослідження від описів окремих випадків до експериментів з групами контролю, і, нарешті, до великомасштабних рандомізованих контрольованих досліджень, яким тепер надається перевага як стандартам наукового доказу.

Разом з тим, методи статистичної обробки отриманих результатів настільки різноманітні, що найчастіше досліднику дуже складно розібратися у них і вибрати адекватні поставленим завданням статистичні критерії. Тривалий час аналіз медичних даних могли здійснювати лише високопрофесійні фахівці, оскільки це вимагало серйозної попередньої підготовки. З появою обчислювальної техніки і вдосконаленням сучасних програм оброблення й аналізу даних статистична обробка піднялася на новий рівень. Тепер дослідник-медик може й не мати математичної підготовки [1].

Досить оперувати статистичними поняттями і, найголовніше, правильно вибрати метод аналізу, володіти навичками роботи на персональному комп'ютері та освоїти новітні пакети програмного забезпечення для статистичного оброблення й аналізу біомедичних досліджень.

# 1 РОЗГЛЯД СУЧАСНОГО ПЗ ДЛЯ ВІЗУАЛІЗАЦІЇ АНАЛІЗУ БІОМЕДИНИХ ДАНИХ. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Аналіз технічного завдання.

За мету поставлено задачу візуального аналізу біомедичних даних, попередньо підготовлених і сформованих у багатоканальному wav файлі. Аналіз здійснюватиметься візуально, за допомогою графіків різного роду, які відображають форму та спектр сигналу до і після виконання фільтрації. Одночасно можна здійснювати аналіз по декільком каналам.

Вимоги до технічних характеристик комплексу.

1. Частота сигналів довільна.
2. Тип вхідних даних : wav файл.
3. В залежності від інформації в заголовку у wav файлі програма повинна оброблювати різну кількість каналів з різною частотою дискретизації та розрядністю представлення даних.
4. Передбачити можливість виведення до 16 різних, одночасних графічних представлень, для кожного з яких має бути можливість вибору типу фільтра.
5. Кожне представлення повинно мати можливість одночасного виведення форми сигналу та його спектру із можливістю масштабування по обом осям кожного графіка.
6. Програма має виконувати фільтрацію різними типами фільтрів в залежності від вибору користувача.
7. Має буди передбачено можливість виведення на графіки амплітуд та спектрів як вхідного так і відфільтрованого сигналів.
8. Програма повинна мати можливість записувати у wav файл відфільтрований сигнал або його частину у заданому часовому проміжку.
9. Програма має мати можливість створювати вкладки для виведення графічної інформації.
10. Кожна вкладка повинна мати можливість задання часового інтервалу.

Також має бути можливість задання одного часового інтервалу для всіх вкладок.

1.2 Розгляд сучасного програмного забезпечення для візуалізації аналізу біомединих даних

В основі оброблення й аналізу даних лежать математичні методи, які за останні півстоліття істотно не змінилися, проте відповідне програмне забезпечення за цей час зазнало істотних змін. Зі зміною поколінь ЕОМ мінялися й програмні засоби обробки та аналізу даних. І якщо можливості перших ЕОМ з аналізу даних не перевершували можливості сучасних середніх калькуляторів, то в 70-і роки вже з'явилися пакети прикладних програм, які містили практично всі ті математичні методи оброблення, які входять до складу й сучасних пакетів. Розвиток пакетів здійснювався шляхом вдосконалювання технології й аналізу (табл. 1).

На сьогоднішній день число пакетів для оброблення інформації досягає кількох десятків, серед яких зарубіжні пакети, такі як: SYSTAT, STATGRAPHICS, BMDP, SPSS, SAS, CSS, Statistica, а також вітчизняні: STADIA, ЕВРІСТА, МЕЗОЗАВР, САНІ, КЛАСС-МАСТЕР, СІГАМД тощо.

Більшість комп'ютерних статистичних програм не є чисто медичними прикладними програмами, оскільки більшість методів статистичного аналізу є універсальними й можуть застосовуватися не лише в різних галузях медичної статистики, але й у найрізноманітніших галузях людської діяльності. Наприклад, з погляду формальної логіки статистичний прогноз інфекційної захворюваності й прогноз курсу долара — це та ж сама задача, а тому вона може вирішуватися за допомогою одних і тих же пакетів прикладних програм.

Основну частину наявних пакетів для оброблення даних можна віднести до трьох категорій: спеціалізовані пакети, пакети загального призначення і професійні пакети.

Спеціалізовані пакети, як правило, містять методи з одного-двох розділів статистики або методи, що використовуються в конкретній предметній галузі (наприклад, Мезозавр — програма аналізу часових рядів). Спеціалізовані пакети застосовуються для вирішення вузького кола завдань з використанням спеціальних методів статистичного аналізу. Експлуатація цих програм вимагає високого рівня підготовки користувача в галузях певних розділів статистики.

Професійні пакети призначені для користувачів, які мають справу із надзвичайно великими обсягами даних або вузькоспеціалізованими методами аналізу. Пакети загального призначення або універсальні (Statgraphics, SPSS, Statistica, Excel, STADIA тощо) є найбільш зручними для користувача-початківця завдяки відсутності орієнтації на специфічну предметну галузь, широкому діапазону статистичних методів і дружньому інтерфейсу користувача. Вони більш доступні для практики й можуть використовуватися широким колом фахівців різного профілю. Практично всі задачі, які стосуються оброблення й аналізу медико-біологічних досліджень, можуть бути вирішені за допомогою універсальних пакетів.

Пакет Statgraphics розроблявся ще для роботи в середовищі DOS, a потім був адаптований до операційної системи Windows і отримав нову назву Statgraphics Plus. Сучасний пакет STATGRAPHICS PLUS - це досить потужна статистична програма, яка містить більше 250 статистичних функцій. За своїми характеристиками пакет займає проміжне місце між SPSS і Statistica.

Пакет SPSS (Statistical Package For Social Science) - це один із найбільш часто використовуваних пакетів статистичної обробки медико-біологічних даних (http://www.spss.com/). Цей пакет створювався ще для «великих» електронно-обчислювальних машин і послідовно переводився для роботи в середовищі DOS, а потім Windows. Пакет SPSS досить потужний і добре відпрацьований, наближається за своїми можливостями до професійних пакетів, і реалізація статистичних процедур добре пристосована до практичної роботи.

Поряд з пакетом SPSS, великою популярністю в освітніх та наукових закладах США користується пакет STATA. Це професійний статистичний програмний пакет з data-management system, який досить часто застосовують для біомедичних цілей.

Програма добре документована, видається спеціальний журнал для користувачів системи. Не менш популярним інструментом розробки користувацьких додатків і не лише в медицині, а й в бізнесі, економіці, фінансах, промисловості тощо є інтегрована система аналізу та управління даними STATISTICA [1].

Пакет Statistica спеціально створювався для роботи в середовищі Windows і відповідає всім стандартам Windows, що дозволяє зробити аналіз високоінтерактивним. STATISTICA складається з набору модулів, в кожному з яких зібрані тематично пов'язані групи процедур, що мають високу швидкість і точність обчислень. Система STATISTICA містить повний набір класичних методів аналізу даних [2]: від основних методів статистики до просунутих методів, що дозволяє гнучко організувати аналіз. Дані системи STATISTICA легко конвертувати в різні бази даних і електронні таблиці. Ця система відрізняється найбільш розвиненим інтерфейсом із користувачем і багатими графічними можливостями, підтримує високоякісну графіку, що дозволяє ефектно візуалізувати дані і проводити графічний аналіз. Гнучка і

потужна технологія доступу до даних дозволяє ефективно працювати як з таблицями даних на локальному диску, так і з віддаленими сховищами даних. Система STATISTICA є відкритою системою містить мови програмування, які дозволяють розширювати систему, запускати її з інших Windows-додатків, наприклад, з Excel.

Електронна таблиця Excel найпоширеніша, оскільки повністю русифікована і найбільш доступна, вона встановлюється автоматично при інсталяції пакета MS Office. Електронна таблиця Excel тісно інтегрується з іншими програмами пакета MS Office, наприклад, MS Word і PowerPoint, а тому найчастіше використовується при оформленні результатів роботи. MS Excel - це електронна таблиця з достатньо потужними математичними можливостями, проте деякі статистичні функції є просто додатковими, а тому розрахунки, зроблені за допомогою вбудованих окремих формул, не завжди визнаються авторитетними біомедичними журналами [4]. MS Excel, як правило, використовується при найпростішому статистичному аналізі даних. Окрім того, в MS Excel неможливо побудувати якісні наукові графіки. MS Excel добре підходить для накопичення даних, попередніх статистичних прикидок, для побудови деяких видів діаграм, проте остаточний статистичний аналіз бажано робити в програмах, які спеціально створені для цих цілей. Існує макрос-додаток XLSTAT-Pro (http://www.xlstat.com) для MS Excel, який включає в себе більше 50 статистичних функцій, включаючи аналіз виживаності.

Програма вітчизняної розробки STADIA включає в себе всі необхідні статистичні функції, призначені для статистичного аналізу даних. Проте ця програма фактично не змінюється з 1996 року, а тому графіки і діаграми, побудовані за допомогою STADIA, виглядають в сучасних презентаціях архаїчно. До позитивних якостей програми можна віднести російськомовний інтерфейс і наявність книг, що описують роботу [3]. Зі сторінки http://www.protein.bio.msu.su/ ~ akula / index.htm можна взяти демо-версію STADIA. Слід зазначити, що всі ці пакети постійно оновлюються і з кожним роком з'являються їх нові версії.

При виборі пакета для аналізу даних можна виділити два аспекти:

а) початковий вибір пакета аналізу;

б) поточний вибір при переході на більш сучасний, більш потужний пакет. Підходи в обох випадках дещо відрізняються.

У першому випадку на вибір накладаються такі обмеження:

1. Можливості комп'ютера;

2. можливості одержання установчої версії пакета;

3. характеристики пакета.

Що стосується першого пункту, то варто вибирати найбільш сучасні версії пакетів із тих, що можуть бути встановлені на наявний комп'ютер. Другий пункт очевидний - вибирати можна з тих пакетів, що доступні. Що стосується характеристик пакета, то тут варто розглянути такі аспекти:

а) обчислювальні можливості;

б) зручність роботи;

в) складність освоєння.

Обчислювальні можливості. У випадку, коли необхідно обробляти медичні дані помірних обсягів (до декількох тисяч спостережень) стандартними статистичними методами, то найкраще використовувати універсальні пакети. Якщо дивитися з позицій лікаря-дослідника, то всі сучасні універсальні статистичні пакети за своїми обчислювальними можливостями повністю відповідають можливим потребам (Statistica, SPSS, Statgraphics Plus та інші пакети, що працюють в операційній системі Windows). Проте завжди варто переконатися, що обраний пакет містить необхідні методи обробки.

Зручність роботи. Всі сучасні пакети досить зручні в роботі (коли вони вже освоєні). Складність освоєння. За складністю освоєння пакети дещо розрізняються і тут варто віддати перевагу русифікованим пакетам або пакетам, з яких є доступна література або є ймовірність пройти курс навчання. Варто зауважити, що без крайньої необхідності (неможливість забезпечити необхідну обробку даних) не бажано змінювати обраний і освоєний пакет аналізу, тому що це призведе до значного збільшення трудовитрат.

Що стосується заміни пакета на більш сучасну версію, то тут є дві крайності:

1. Прагнення до постійного відновлення, установки самих останніх версій пакетів, як правило, віднімає багато сил, не дозволяє виробитися корисним стереотипам дій, у той же час не приводить до суттєвого зростання можливостей.

2. З іншого боку, уподобання застарілих пакетів найчастіше не дозволяє повною мірою використовувати можливості сучасної техніки і програмного забезпечення. Існує деякий емпіричний оптимум, що може визначатися зразковим терміном експлуатації пакета в 2-3 роки, після закінчення котрого доцільно здійснювати перехід до більш сучасних пакетів. При цьому переважно вибирати чергову версію того ж пакета, що використовувався раніше. Запам'ятайте: спадкоємність значно полегшує процес освоєння пакета.

Висновок

Широке впровадження сучасних комп'ютерних технологій і застосування пакетів прикладних програм докорінно змінило процес оброблення й аналізу медичних даних. Завдяки використанню комп' ютерів і сучасного програмного забезпечення, оброблення й аналіз медико-біологічних даних став набагато легшим. При цьому для застосування основних статистичних методів оброблення медичних даних лікарю не потрібно заглиблюватися в складність математичних процедур, а варто зрозуміти, для чого і як ці методи використовуються, а також вміло використовувати обраний пакет прикладних програм.

# ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Процес проектування складного програмного забезпечення починається з уточнення його структури, тобто визначення структурних компонентів і зв'язків між ними. Тому в даному розділі слід розглянути сучасні принципи проектування ПЗ, а також спроектувати програмне забезпечення для візуалізації біомедичних сигналів, як на фізичному так і логічному рівні. Результатом проектування має стати набір діаграм які розкривають можливості для користувача, і демонструють компоненти, які потрібні для розробки та реалізації потрібного функціоналу.

2.1 Діаграма варіантів використання (use case diagram)

Призначення даної діаграми полягає в наступному: проектована програмна система представляється у формі так званих варіантів використання, з якими взаємодіють зовнішні сутності або актори. При цьому актором або дійовою особою називається будь-який об'єкт, суб'єкт або система, що взаємодіє з модельовану бізнес-системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на модельовану систему так, як визначить розробник.

Варіант використання служить для опису сервісів, які система надає актору. Іншими словами кожен варіант використання визначає набір дій, які виконуються системою при роботі з актором. При цьому не важливо як це буде реалізовано.

Розглядаючи діаграму варіантів використання в якості моделі бізнес-системи, можна асоціювати її з "чорним ящиком". Концептуальний характер цієї діаграми проявляється в тому, що докладна деталізація діаграми або включення в неї елементів фізичного рівня уявлення на початковому етапі проектування швидше має негативний характер, оскільки визначає способи реалізації поведінки системи. Ці аспекти повинні бути свідомо приховані на діаграмі варіантів використання. Тому будуть розглядатися основні вимоги до програмного майбутнього програмного продукту, без розгляду їх реалізації [6].

Розробка продукту передбачає створення зручного багатофункціонального комплексного додатку для аналізу вхідного багатоканального сигналу, який можна було використовувати в не лише в медичних цілях. Програма повинна мати велику кількість вбудованих інструментів для візуалізації вхідного сигналу, його модифікації, а саме головне порівняння та аналізу вихідних даних. Крім того має бути можливість розширення функціоналу по мірі потреб, що потребувало написання добре підтримуємого коду.

Отже основним користувачем звісно виступає особа, яка саме і бере участь у процесі аналізу вхідного сигналу. Вона має отримати доступ до великої кількості інструментів, які мають бути представленні зручним користувацьким інтерфейсом. Крім того кожен інструмент повинен бути не лише простий у використанні, а й максимально конфігуруватися як користувачем під час роботи так і програмістом з подальшим розвитком продукту [7].

Ці інструменти можна розбити на категорії повезені з призначенням:

* інструменти керування сценою:

1. створення нових представлень;
2. редагування та видалення уже створених;

В то самий час окрім особи яка проводить аналіз, програма може використовуватися програмістами для відлагодження функціоналу програми, за допомогою системи логування. Згідно проаналізованого матеріалу було побудовано, UML діаграму варіантів використання, яка наведена на рисунку 1.1.



Рисунок 1.1 – Діаграма варіантів використання

* 1. Основні компоненти. Діаграма компонентів (component diagram)

Для створення конкретної фізичної системи необхідно реалізувати всі елементи логічного представлення в конкретні матеріальні сутності. Для опису таких реальних сутностей призначений інший аспект модельного уявлення, а саме - фізичне представлення моделі. У контексті мови UML це означає сукупність пов'язаних фізичних сутностей, включаючи програмне і апаратне забезпечення, а також персонал, які організовані для виконання спеціальних завдань.

Для реалізації системи необхідно розробити вихідний текст програми на мові програмування. При цьому вже в тексті програми передбачається організація програмного коду, що визначається синтаксисом мови програмування і передбачає розбиття вихідного коду на окремі модулі.

Однак вихідні тексти програми ще не є остаточною реалізацією проекту, хоча і служать фрагментом його фізичного представлення. Програмна система може вважатися реалізованою в тому випадку, коли вона буде здатна виконувати функції свого цільового призначення. А це можливо, тільки якщо програмний код системи буде реалізований у формі виконуваних модулів, бібліотек класів і процедур, стандартних графічних інтерфейсів, файлів баз даних. Саме ці компоненти є базовими елементами фізичного представлення системи в нотації мови UML.

Особливістю програмного забезпечення, що розроблється, є його функціональна навантаженість. Така форма передбачає використання великої кількості інструментів, а також складних елементів користувацького інтерфейсу, що утворюють окремі модулі із власними залежностями один від одного. Зв'язок між модулями має бути тісний, але частина з них маж бути більш незалежна для того щоб можна було додавати новий функціонал, без серйозних змін в інших компонентах [7].

Оскільки програма має працювати з біомедичними сигналами представленні у бігатопотокову аудіо файлі, то варто використовувати бібліотеку або ряд бібліотек для стандартизованого аналізу аудіо потоку, а також для отримання спектру. Для уникнення проблем пов’язаних з типізацією даних та використання з іншими бібліотека, слід створити обгортку на вибрану бібліотеку.

Для виводу графіки на екран варто використовувати бібліотеку яка б дозволила використовувати апаратні ресурси машини для виводу графіки (особливо відеоадаптер). Це дозволить розвантажити центральний процесор і дасть розширити функціонал програми без втрати швидкодії. Як із попереднюю бібліотеко слід обгорнути бібліотеку, а також розробити додаткові модулі для виводу складних графічних елементів, приведених до об’єктів [6].

Оскільки програма матиме досить складний і багатий візуальний інтерфейс слід вибрати framework (набір бібліотек), який не лише дозволив використовувати стандартні елементи користувацького інтерфейсу (user interface UI), а й спростити роботу з мовою програмування і дати можливість створювати власні елементи UI за допомогою одної із багатьох мов розмітки та задання стилю.

Крім вище сказано потрібно щоб вибраний framework мав можливість роботи з різними типами даних. Не менш важливим моментом є можливість переносу коду на інші платформи і максимальна інтегрованість об’єктних принципів програмування.

Згідно проаналізованого матеріалу та технічного завдання було побудовано діаграму компонентів зображену на рисунку 1.2.



Рисунок 1.2 – Діаграма компонентів

Розглянемо основні представлені блоки, як окремі, автономні одиниці.

* + 1. Головне вікно

Головне вікно має представлятися як ключовий елемент користувацького інтерфейсу. Він має давати доступ до більшості інструментів програми. Сам елемент виконує роль контейнера для візуального інтерфейсу, що має бути підготовленим UI Engine. Такий підхід дозволить позбавитися зайвих залежностей і створювати незалежні модальні вікна із простим користувацьким інтерфейсом, що не лише спростить розробку, а й прискорить її.

* + 1. UI Engine та Templates

Інтерфейс користувача майбутнього додатку, є не стандартним і тому є потреба в розробці власних елементів користувацького інтерфейсу, за допомогою специфічної мови розмітки, які мають знімати обмеження щодо відображення та керування. Не менш важливим є використання апаратного прискорення для відображення усіх складових інтерфейсу.

Для цього потрібно сотворити або розробити власний модуль відображення (UI Engine) який відповідав усім вимогам наведених вище. Крім того необхідна повна підтримка динамічної зміни абсолютно усіх елементів сцени, а також можливість відділити програмний код користувацького інтерфейсу від коду обробки даних. Такий модуль повинен мати змогу працювати з так званими шаблонами (Templates), це дозволить уникнути повторювання коду, а також надасть можливість уніфікувати зовнішній вигляд майбутнього продукту.

* + 1. Spectrum Engine

Даний модуль виконуватиме зчитування та обробку вхідного потоку даних, його відтворення на вибраний аудіо пристрій, а також підготовлюватиме дані для візуалізації, діаграма зображена на рисунку 1.3.

Модуль має містити компоненти які б дозволили йому, опрацьовувати вхідний багатоканальний wav файл, зчитуючи формат даних із заголовку файлу, а також створювати вихідні файли після обробки.



Рисунок 1.3 – Діаграма Spectrum Engine

Модуль має підготовлювати дані для побудови графіків спектру та форми сигналу, для цього потрібно розробити під-модулі WaveForm Visulalisation та Spectrum Visulalisation. Так як побудувати спектр в певному діапазоні та з певною точністю, не легка задача Spectrum Visulalisation тісно пов'язаний з модулем Spectrum Analyzation, який в свою чергу використовує бібліотеку для швидкого перетворення Фур’є.

Оскільки програма повинна обробляти та відображати декілька потоків одночасно, то підготовка даних має відбуватися в окремих потокох, а також мати зручний сервіс для створення нових об’єктів, які б містили всі необхідні дані для відображення, в ролі такого сервісу виступає Graph Service. Даний сервіс дозволить конфігурувати набір фільтрів які будуть накладатися на вхідний потік даних. Таким чином можна буде створити декілька графіків, представлені об’єктами Graph, які б використовували одні й ті ж дані, а не дублювали їх.

# РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі виконується вибір мови та бібліотек, необхідність яких обґрунтована в попередньому розділі. Також у розділі описано розробку та тестування таких ключових модулів програми як: системи відтворення та обробка аудіо, систему відображення, а також системи управління сценою, тобто її контентом.

3.1 Обґрунтування вибору мови програмування

Реалізація системи обумовлена технічним завданням. Для написання програми необхідно проаналізувати і обрати мову, що врахує усі вимоги до мови, обґрунтування яких було здійснено під час проектування програмного продукту.

Згідно проаналізованого матеріалу у попередньому розділі, існує потреба у використання набору бібліотек, які б полегшили написання користувацького інтерфейсу, а також надали можливість написання власних елементів UI. Саме тому при виборі мови варто звернути увагу не лише переваги мови, а й на можливості найпоширеніших фреймворків для даних мов, а також їх використання у корпоративних рішеннях.

3.1.1 Вибір мови програмування

С++ – стандартизована об'єктно-орієнтована мова програмування. Вона є ефективною для створення системного програмного забезпечення, та прикладних програм. Має зручні механізми управління пам’яттю та даними. Крім того має велику базу підготовленого коду (бібліотек, фреймворків, тощо). Швидкодія програм написаних на С++ дуже висока [8].

Для вибору розглянуто такі критерії, як наявність об’єкто-орієнтовної парадигми, застосування багатопоточності, багатоплатформеність, швидкість виконання, оскільки вони є найактуальнішими при виборі мови. Не менш важливим є і кількість підготовленого коду. З таблиці 3.1 можна зробити висновок, що найкращою мовою для реалізації даного продукту є С++, оскільки ця мова має більше переваг ніж інші.

Таблиця 2.1 – Порівняння сучасних мов програмування

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Можливості | Assembler | C++ | C# | PHP | Java |
| Об’єктно-орієнтована | – | + | + | + | + |
| Динамічні масиви | – | + | + | + | + |
| Багатопоточність | – | + | + | + | + |
| Узагальнене програмування | – | + | + | + | + |
| Багатоплатформеність | – | + | +/– | + | + |
| Статична типізація | – | + | + | – | + |
| Швидкість виконання | + | + | + | +/– | +/– |

3.1.2 Вибір бібліотеки класів

Qt - це крос-платформна бібліотека C ++ класів для створення графічних користувацьких інтерфейсів (GUI). Ця бібліотека повністю об'єктно-орієнтована, що забезпечує легке розширення можливостей і створення нових компонентів. Вона підтримує величезну кількість платформ, таких як Windows, Linux, Sun Solaris, HP-UX, Digital Unix, IBM AIX, SGI IRIX і багато інших. З моменту першої появи комерційної версії Qt в 1996 році з'явилася кілька сотень відомих по всьому світу додатків.

Для бібліотеки створення GUI об'єктно-орієнтована мова є єдино придатним засобом, а стандартна об'єктна модель C ++ забезпечує ефективну і швидку розробку програм, нарощування необмежених можливостей і швидку модернізацію. Природно через це в якості базової мови для бібліотеки обраний C ++. Але на додаток до можливостей самої мови в бібліотеці Qt додані декілька хороших можливостей:

* потужний механізм комунікації між об’єктами;
* механізм створення властивостей об'єктів;
* підтримку подій і фільтрів подій;
* переклад рядків для підтримки інтернаціоналізації;
* підтримку внутрішніх таймерів, які дозволяють інтегрувати багато завдань для складних GUI;
* ієрархічні дерева об'єктів, які є свого роду генеалогічними деревами.
* "безпечні" вказівники, які автоматично встановлюються в NULL при видаленні об'єкта, на який веде посилання.

Qt Quick - новий перспективний інструмент Qt, що дозволяє проектувати UI додатки на простому javascript-подібній декларативній мові (QML), використовуючи всі можливості Qt Animation Framework і навіть більше. Завдяки Qt Quick, створення UI і коду стають повністю незалежними етапами розробки. Qt Quick відкриває нові можливості при переносі на інші платформи і кастомізації додатків, скорочує терміни розробки і виводить UI на якісно новий рівень [9].

Всі ці можливості реалізовані у вигляді спеціальних класів C ++. Поява нових сучасних вимог до призначених для користувача інтерфейсів тягне за собою і появу нових нестандартних механізмів, які сама мова програмування забезпечити не здатний, а лише після створення спеціалізованих класів.

Найбільш привабливим рішенням згідно проаналізованого матеріалу є використання бібліотеки класів Qt оскільки вона дає використовувати всі переваги мови С++, яка є найшвидшою хоча і не самою зручною із розглянутих мов програмування. Вибрана платформа є не новою, тому її надійність значно спрощує розробку програмного продукту, а інструмент Qt Quick та мова QML надають можливість створити складний користувацький інтерфейс, як незалежну одиницю.

Не менш важливим критерієм є те що бібліотека підтримує, велику кількість платформ для розгортання з максимальною адаптивністю до звичних компонентів, а також безкоштовне ліцензування більшості компонентів для корпоративних рішень.

Qt Creator — інтегроване середовище розробки, призначене для створення крос-платформових застосунків з використанням бібліотеки Qt. Підтримується розробка як класичних програм мовою C++, так і використання мови QML, для визначення сценаріїв, в якій використовується JavaScript, а структура і параметри елементів інтерфейсу задаються CSS-подібними блоками. Qt Creator може використовувати GCC або Microsoft VC++ як компілятор і GDB як зневаджувач. Для Windows версій бібліотека комплектується компілятором, заголовними і об'єктними файлами MinGW.

Щоб розпочати інсталяцію Qt, пройдіть по цьому посиланню - http://download.qt-project.org/official\_releases/online\_installers. Ви побачите список настановних файлів і вам належить вибрати той, який підходить для вашої операційної системи.

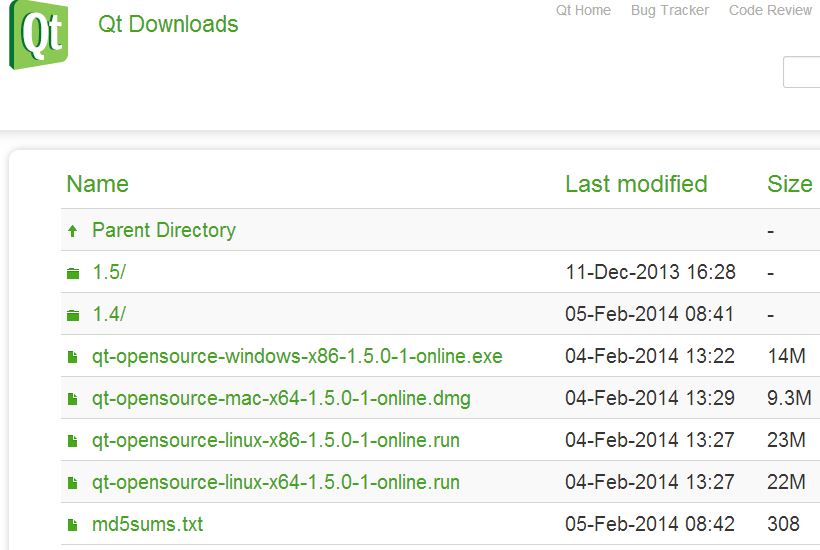


Рисунок 2.1 – Варіанти установників під різноманітні платформи

Після чого слід виконати звичайний запуск відповідного виконавчого файлу в залежності від вибраної операційної системи. Під час установки слід вказати директорію де буде розміщуватися бібліотеки та ПЗ для розробки. Після чого вибрати набір бібліотек які варто установити.

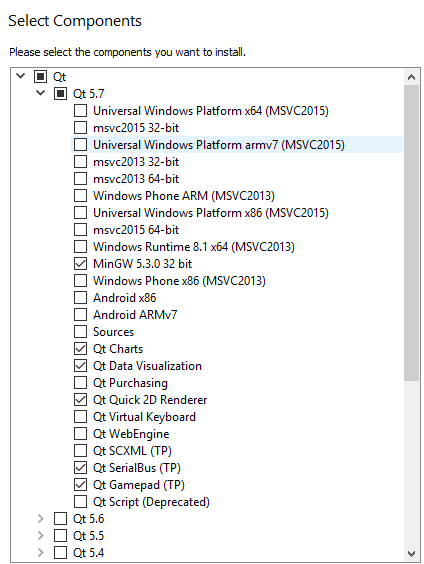
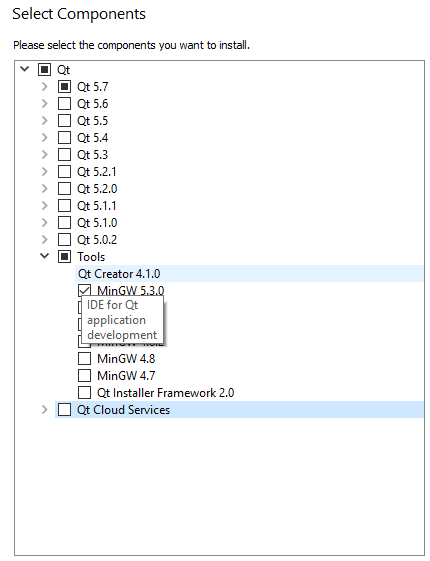


Рисунок 2.2 – Конфігурування процесу установки

Мінімальна версія Qt 5.5, мінімальний набір модулів наведені на рисунку. Версія компілятора та тип бібліотеки залежить від платформи на якій відбуватиметься подальший запуск. Тестування та розробка проводиться на версії 5.7 MinGW 530.32.

3.2 Реалізація головного вікна

Головне вікно побудоване на основі класу QMainWindow. Даний клас дозволяє отримати доступ до стандартних елементів користувацького інтерфейсу, ініціалізація даного класу наведена нище:

MainWidget::MainWidget(QWidget \*parent):QMainWindow(parent), m\_mode(NoMode), m\_engine(new Engine(this)), m\_progressBar(new ProgressBar(this)), m\_spectrograph(new Spectrograph(this)),

m\_levelMeter(new LevelMeter(this)), ui(new Ui::MainWindow),

m\_settingsDialog(new SettingsDialog(m\_engine->availableAudioInputDevices(),

m\_engine->availableAudioOutputDevices(), this))

{

m\_spectrograph->setParams(SpectrumNumBands, SpectrumLowFreq, SpectrumHighFreq);

createUi();

connectUi();

}

Під час ініціалізації відбувається створення елементів користувацького інтерфейсу, налаштування масштабування, стилів та створення з’єднання між елементами UI. Прикладом такого з’єднання є створення реакції зміну формату вхідного файлу:

CHECKED\_CONNECT(m\_engine, SIGNAL(formatChanged(const QAudioFormat &)), this, SLOT(formatChanged(const QAudioFormat &)));

Головними компонентами вікна є QQuickWidget та SettingsDialog. Перший виконує роль UI Engine описаного раніше і відповідає за відображення усіх елементів UI, які відповідають за створення та налаштування графіків, а також їх відтворення в реальному часі. Другий компонент є модальним вікном з налаштуванням вікна для спектрального аналізу та пристроїв виведення. Вікно зконфігуровано, для роботи як у віконному так і повноекранному режимі.

QQuickWidget дозволяє не лише відображати підготовлені шаблони, а й створити зв'язок між основною логікою програмного продукту та логікою користувацького інтерфейсу. Самим головним моментом є не прямий зв'язок, а на основі інтерфейсів. Основною одиницею в UI є QML елементи, розташування яких наведена на рисунку 2.3.

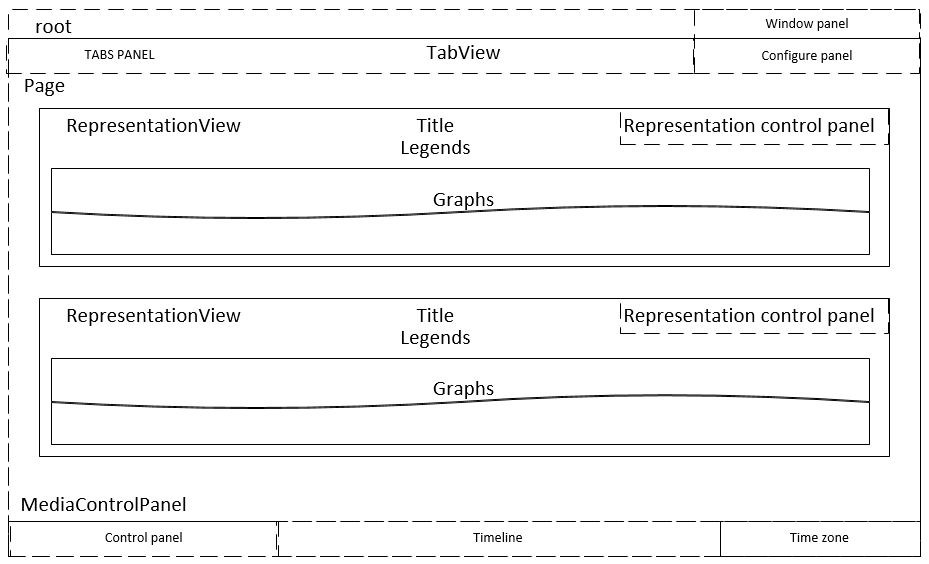


Рисунок 2.3 – Розташування QML елементів на сцені

Розглянемо кожний елемент окремо:

1. Root – головний контейнер, який містить усі елементи UI, а також ряд констант для їх налаштувань;
2. Window panel – містить набір кнопок та функцій для переходу у повноекранний режим роботи та особливостей роботи в ньому;
3. TabView – контейнер який містить усі створені робочі вкладки, а також їх контент;
   1. Tabs Panel – містить набір кнопок які дозволяють перейти на створену вкладку, видалити цільову вкладку або створити нову;
   2. Page – являє собою контент вибраної вкладки;
   3. RepresentationView – є контейнером для представлення, який місить заголовок та легенду поточного представлення;
   4. Representation control panel – дає можливість видалити поточне представлення, а також зконфігурувати його.
4. Configure panel – містить набір кнопок, що дають доступ до конфігурування пристроїв виводу, вікон спектрального аналізу та можливість створювати нові представлення у поточній вкладці;
5. Media Control Panel – є контейнером для панелей керуванням медіа контентом.
   1. Control panel – містить кнопки та функції для відкриття файлу для опрацювання і управлінням процесом відтворення;
   2. Timeline – відображає поточний час відтворення у процентному співвідношенні, а також дозволяє візуально змінити поточний час відтворення;
   3. Time zone – являє собою цифровим варіантом Timeline.

Кожен елемент є автономний та написаний з нуля, що дозволяє досягнути бажаного функціоналу та поведінки створеного елементу. Весь інтерфейс спроектовано в строгому та набравшому популярності дизайні у темних тонах для максимальної зручності в роботі, а використання елементів анімації дозволи зробити роботу з програмою більш плавною та зрозумілою. Для підготовки до друку інтерфейс переводиться у світлі тони, а роздільна здатність вихідного зображення зростає до 600dpi. Узагальнений приклад такого елементу, а саме панелі відображення поточного часу відтворення та його зміни, наведено нище:

Row{

TextInput{

id: hours

} Text {

id: hoursTitle

} TextInput{

id: minutes

} Text {

id: minutesTitle

} TextInput{

id: seconds

}

}

Хоча кожен елемент є автономною одиницею, він все одно реагує на зміні у середовищі за вдяки своєму батьку, наприклад вище згадана панель реагує на зміну поточного часу за допомогою “Timeline”, у якого з ним спільний батько. Зовнішній вигляд елементів наведено на рисунку 2.4.

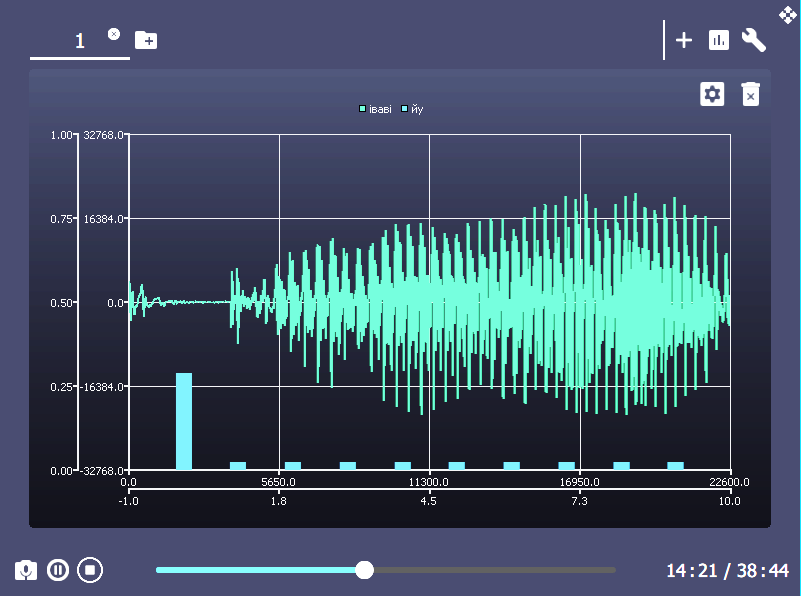


Рисунок 2.4 – Зовнішній вигляд підготовленої сцени

Крім видимих елементів UI, є безліч спливаючих вікон, які дозволяють конфігурувати фільтри та графіки. Організація обміну даних між С++ та QML здійснюється інтегрованими засобами Qt.

3.3 Реалізація Spectrum Engine, класів обробки та відтворення   
вхідного потоку

Клас організований на основі основного класу QObject. Наслідування дає можливість використовувати внутрішню систему обміну повідомленнями, схожу на ту що використовується для обміну повідомленнями між елементами UI. Клас містить поля та функції для доступу до пристроїв вводу/виводу звуку, а також для управління потоком даних (старт/зупинка/пауза відтворення, зміна позиції відтворення):

const QList<QAudioDeviceInfo> &availableAudioOutputDevices() const

{ return m\_availableAudioOutputDevices; }

bool loadFile(const QString &fileName);

void startRecording();

void startPlayback();

void suspend();

void stopPlayback();

void setTimePosition(qint64 time);

Увесь час представлений у вигляді мікросекунд, що дає можливість точно задати проміжок часу для відтворення, для зберігання використовується 64 біт. Даний клас виступає контролером для решти класів. Він повідомляє решту класу про зміну формату вхідних даних, позиції відтворення, зміну статусу відтворення і т.д., але не здійснює безпосередню обробку сигналу. Повідомлення відсилаються за допомогою так званих сигналів (signals), головні з яких наведені нище:

signals:

void stateChanged(QAudio::Mode mode, QAudio::State state);

void infoMessage(const QString &message, int durationMs);

void errorMessage(const QString &heading, const QString &detail);

void formatChanged(const QAudioFormat &format);

void playPositionChanged(qint64 position);

void bufferChanged(qint64 position, qint64 length, const QByteArray &buffer);

Зчитування даних відбувається по таймеру у два потоки, що забезпечує отримання даних як для візуалізації так і для звукового супроводу. Після того як чергова порція даних зчиталися, про це повідомляється за допомого сигналу bufferChanged. Для зчитування даних з файлу потрібно отримати інформацію про канали даних, що містяться у файлі який аналізується. Для отримання даних про потоки потрібно прочитати заголовок файлу, в якому присутня інформація про розмір заголовка, кількість каналів, частота дискретизації, кількість біт на «семпл», розташування старшого біту і т.д.. Такий функціонал реалізовує клас WavFile наслідуваний від класу QFile.

Клас містить інформацію про доступні фільтри, а також зберігає і дозволяє створювати нові GraphFilterService, які здійснюють обробку потоку даних отриманих від Spectrum Engine. GraphFilterService являє собою набір фільтрів, а також модулів для підготовки вихідних даних для відображення графіків на представленнях. Кожен сервіс може формувати дані, для декількох представлень. Робота даного сервісу зображено на рисунку 2.5.



Рисунок 2.5 – Алгоритм роботи GraphFilterService

Для того щоб, графік представлення мав доступ до даних сервісу, його слід підписати на розсилку нових значень, за допомогою методів subscribeWaveForm та subscribeSpectrum. Дані методи приймають адресу масиву з точками певного графіку, а також номер каналу з якого слід отримувати дані. Після виконання підписки сервіс автоматично обновлює стан графіків в залежності від типу графіку. Якщо графік видаляється слід виконати обернену операцію до підписки за допомогою методів unsubscribeWaveForm та unsubscribeSpectrum.

Фільтрація виконується за допомогою методу doFilter класу що реалізує інтерфейс Filter. Існує два види фільтрів, це NullFilter та CustomFilter. Перший не виконує ніяких операцій на даними і дає можливість отримати прозорі дані. Другий є фільтром 128 порядку параметром якого є вхідна комбінація 128 параметрів фільтру. Підхід з прозорим фільтром дозволяє використовувати єдиний механізм для відтворення як фільтрованих так і не фільтрованих даних. Для отримання даних спектру використовується три класи SpectrumAnalyser, SpectrumAnalyserThread та Spectrograph.

SpectrumAnalyser, SpectrumAnalyserThread класи які відповідають за обрахунок спектру в окремому потоці для кожного набору фільтрів. Основними методами є calculate, calculateSpectrum відповідно. Метод сalculateSpectrum приймає масив семплів, а також вхідну частоту, та розмір семпла. Алгорим роботи методу наведений на рисунку 2.6.



Рисунок 2.6 – Алгоритм обрахунку та підготовки даних спектру

Після отримання значень амплітуд та спектру, вони передаються об’єктам класів WaveFormCustom та Spectrograph. Класи опрацьовують вхідних потік даних та підготовлюють їх до виводу на екран. Обидва класи працюють з даними у форматі QXYSeries. Даний формат представляє собою координати X та Y. Код підготовки даних спектрального аналізу наведено нижче:

m\_bars.fill(Bar());

FrequencySpectrum::const\_iterator i = m\_spectrum.begin();

const FrequencySpectrum::const\_iterator end = m\_spectrum.end();

for ( ; i != end; ++i) {

const FrequencySpectrum::Element e = \*i;

if (e.frequency >= m\_lowFreq && e.frequency < m\_highFreq) {

Bar &bar = m\_bars[barIndex(e.frequency)];

bar.value = qMax(bar.value, e.amplitude);

bar.clipped |= e.clipped;

}

}

const int numBars = m\_bars.count();

for(int j = 0; j < subscribtionList.size(); j++)

{

subscribtionList[j]->clear();

for (int i=0; i<numBars; ++i) {

for (int var = 0; var <= qFloor(m\_bars[i].value\*50); var++) {

subscribtionList[j]->append(QPointF(i, (float)var/50));

}

subscribtionList[j]->append(QPointF(i, m\_bars[i].value));

}

Обновлення даних UI реалізовано на основі так званих підписок. Будь яке представлення має змогу підписатися на отримання даних з того чи іншого GraphFilterService, при чому або спектру, або форми сигналу, або одночасно обох. Такий підхід дає змогу з економити ресурси ЦП, оскільки підготовка даних здійснюється однин раз, але отримувати дані можуть зразу декілька графіків.

2.4 Реалізація динамічної роботи інтерфейсу та обробки даних

Головною метою програмного продукту є не лише візуалізація вхідного сигналу, а й порівняння одночасно декілька потоків даних з різними методами обробки у вибраний проміжок часу. Для цього створено користувацький інтерфейс який надає таку можливість.

Кожне представлення являє собою набір графіків, кожен з яких має власну назву, а також осі для відображення даних. Зображення декількох одночасних графіків та візуальний вигляд представлення зображено на рисунку 2.7.

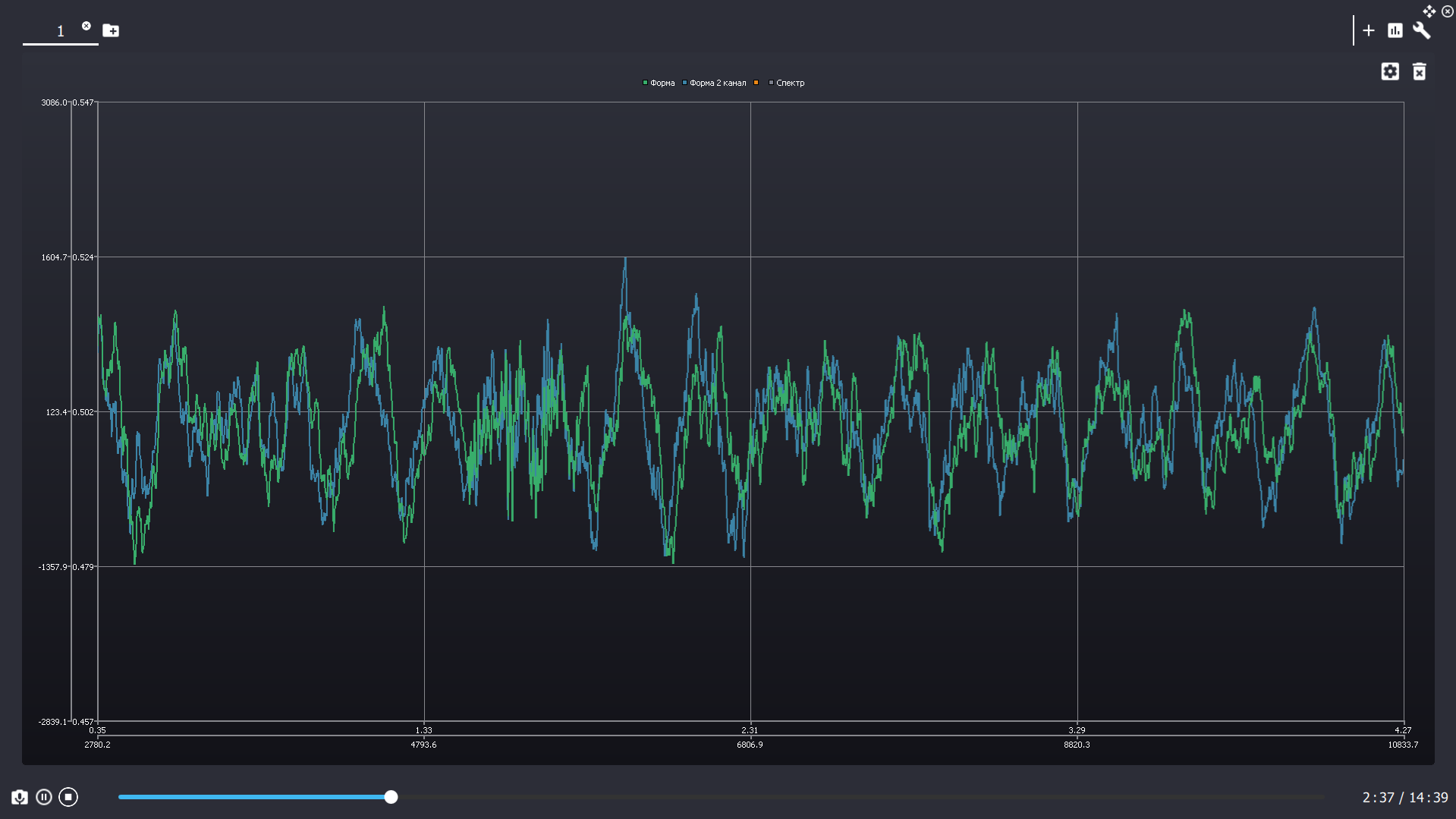


Рисунок 2.7 – Зображення декількох каналів одночасно

Цей набір графіків конфігурується за допомогою діалогового вікна, за виклик якого відповідає кнопка на панелі керування представленням. Зображено на рисунку 2.8.

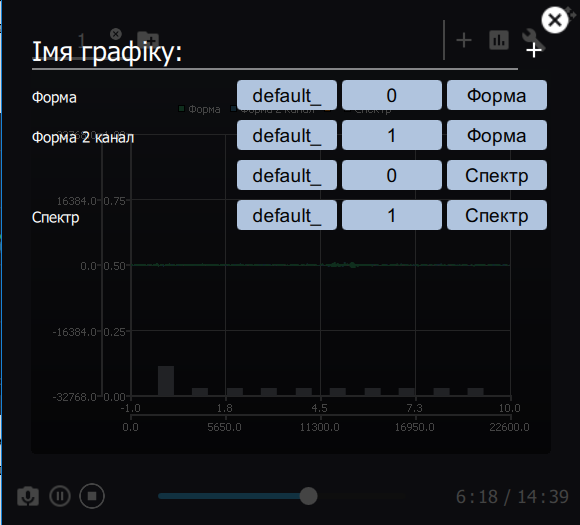


Рисунок 2.8 – Конфігурування графіків представлення

У даному вікні є можливість додати, змінити або видалити графік з вибраного представлення. При створенні графіку слід вказати його назву, після чого вибрати один з наборів доступних фільтрів, доступний канал та який тип даних ми хочемо відобразити (форма чи спектр). При редагуванні можна змінити усі параметри.

Інтерфейс представлення та його налаштування описаний у ScopeView. Основними компонентами є ChartView, ModalWindowGraphsControll, а також елементи графіку ScatterSeries, LineSeries, ValueAxis [10].

Елемент ChartView є вбудованим елементом QML і надає можливість створювати графіки на основі таких елементів PieSeries, LineSeries, SplineSeries, AreaSeries, ScatterSeries, BarSeries, StackedBarSeries, PercentBarSeries, HorizontalBarSeries і т.д.. Такі елементи дозволяють легко побудувати графік будь-якого типу, але лише ScatterSeries, LineSeries мають можливість апаратного прискорення для відображення, що надає змогу змінювати параметри графіку в режимі реального часу. Тому саме на основі цих елементів побудовані власні елементи (альтернативи BarSeries та StackedBarSeries), що забезпечують високу швидкодію. Елемент ModalWindowGraphsControll являє собою раніше згадане модальне вікно з налаштуваннями графіків у предаставленні.

Не менш важливим елементом ScopeView є MouseArea – вбудований елемент QML, що дає можливість відслідкувати поведінку мишки або сенсору в його межах. Саме на основі даного елемента побудована система масштабування. Масштабування реалізоване як по певній області так і у вигляді так званого променю. Таким чином можна як швидко переглянути сигнал як у певній ділянці так і повільно розглядати, певний, точковий об’єкт. Приклад такого масштабування наведено на рисунку 2.9.

Основними методами даного елементу є update та сlear. Перший виконує перебудовую представлення після внесення змін у його структуру у діалоговому вікні ModalWindowGraphsControll, і викликається після

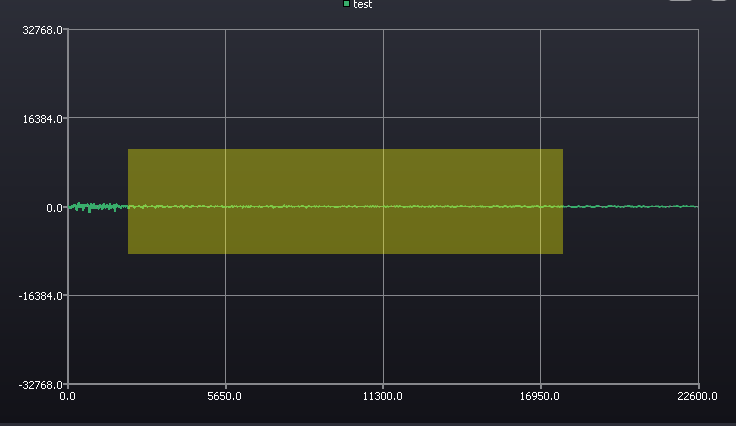


Рисунок 2.9 – Зображення процесу масштабування по вибраній ділянці

завершення роботи з ним. Другий виконує роль ліквідатора, і має місце у першому. Робота методу update наведено на рисунку 2.10



Рисунок 2.10 – Алгоритм обновлення

На основі елементів ScopeView побудований елемент Page. Даний елемент виступає в ролі контейнера який, містить в собі набір таких предствалень. Основними його методами є removeView та createNewView. Особливістю даного контейнера, це те що він автоматично розташовує представлення на сцені, а також налаштовує їх висоту. У свою чергу TabView складається з набору Page-ів, а також наборів інструментів, що дозволяють налаштовувати вкладки, а також конфігурувати набір фільтрів, перейти в налаштування програми і додавати нові представлення.

Крім того було додано можливість виведення відфільтрованих даних у вихідний файл у певному діапазоні, а також прослуховування та аналіз відфільтрованих даних в режимі реального часу. Інтерфейс налаштування поточного фільтру наведено на рисунку 2.11.

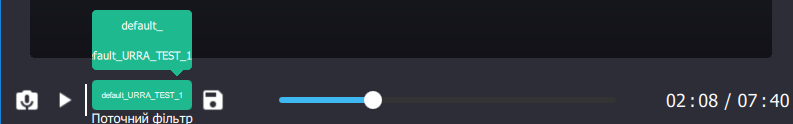


Рисунок 2.11 – Вибір поточного фільтру

Не менш важливим елементом є вікно налаштування, яке дозволяє вибрати пристрої для введення ти виведення звуку, а також встановлення «вікна».

# ВИСНОВКИ

У курсовій роботі виконано проектування та розробка комплексного програмного забезпечення для звукового та візуального аналізу, а також фільтрацію вхідних біомедичних даних.

В результаті аналізу технічного завдання було визначено його суть, мета та призначення.

Проаналізовано сучасний стан програмного забезпечення в галузі аналізу біомедичних сигналів. Зроблено аналіз сучасних продуктів, виділені їхні недоліки та переваги, зроблені висновки.

Розглянуто особливості проектування програмного забезпечення з використанням об’єктно орієнтованого підходу за допомогою універсальної мови проектування UML. Побудовано діаграму використання, що відображає можливості користувача, а також ряд діаграм для визначення необхідного набору вимог та інструментів для розробки програмного продукту.

Обґрунтовано вибір мови програмування, середовища розробки та набір бібліотек, які полегшили написання програмного продукту. Використовуючи вибрані бібліотеки розроблені різноманітні модулі програми для, відображення та формування аудіо контенту, створення та відображення фільтрів у вигляді спектру та форми сигналу. Розробка таких модулів надала можливість створювати власні та використовувати уже створені елементи інтерфейсу, що зробили роботу з програмою більш зручною та зрозумілою.

# ЛІТЕРАТУРА

1. Боровиков В. Statistica. Искусство анализа данных на компьютере / В. Боровиков. - СПб.: Питер, 2001. - 656 с.
2. Гойко О. В. Практичне використання пакета STATISTICA для аналізу медико-біологічних даних : навчальний посібник для студентів вищих навчальних закладів / О. В. Гойко. - Київ, 2004. - 76 с.
3. Кулаічев А. П. Методи і засоби аналізу даних у середо- вищі Windows /А. П. Кулаічев. - М. : Інко, 2002. - 341 с.
4. Лапач С. Н. Статистические методы в медико-биологи- ческих исследованиях с использованием Excel / С. Н. Ла- пач, А. В. Чубенко, П. Н. Бабич. - Издательство "Морион Лтд", 2000. - 320 с.
5. Мінцер О. П. Оброблення клінічних і експерименталь- них даних у медицині : навч. посібник / О. П. Мінцер, Ю. В. Вороненко, В. В. Власов. - К. : Вища школа, 2003. - 350 с.
6. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS / С. Орлов. – 2-е изд. – СПб.: Питер, 2006. — 736 с. — [ISBN 5-46900-599-2](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5469005992).
7. Крэг Ларман Применение UML 2.0 и шаблонов проектирования – 3-е изд. / Крэг Ларман – М.: Вильямс, 2006. — 736 с. — ISBN 0-13-148906-2.
8. Семеренко В. П. Системне програмне забезпечення. Системне об’єктно-орієнтоване програмування мовою С++. Курсове проектування. Самостійна та індивідуальна робота студентів / В. П. Семеренко. – Вінниця : ВНТУ, 2015. – 195 с.
9. Макс Шлее Qt 5.3 Профессиональное программирование на C++. / Шлее Макс – СПб.: «БХВ-Петербург», 2015. – с. 928. – ISBN 978-5-9775-3346-1.
10. Qt Charts : [Електронний ресурс] – режим доступу до ресурсу : http://doc.qt.io/qt-5/qml-qtcharts-chartview.html

# ДОДАТКИ

ДОДАТОК A

Код головного вікна С++

#include "engine.h"

#include "levelmeter.h"

#include "mainwidget.h"

#include "waveform.h"

#include "progressbar.h"

#include "settingsdialog.h"

#include "spectrograph.h"

#include "utils.h"

#include <QLabel>

#include <QPushButton>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QStyle>

#include <QMenu>

#include <QFileDialog>

#include <QTimerEvent>

#include <QMessageBox>

const int NullTimerId = -1;

MainWidget::MainWidget(QWidget \*parent)

: QMainWindow(parent)

, m\_mode(NoMode)

, m\_engine(new Engine(this))

#ifndef DISABLE\_WAVEFORM

, m\_waveform(new Waveform(this))

#endif

, m\_progressBar(new ProgressBar(this))

, m\_spectrograph(new Spectrograph(this))

, m\_levelMeter(new LevelMeter(this))

, ui(new Ui::MainWindow)

, m\_settingsDialog(new SettingsDialog(

m\_engine->availableAudioInputDevices(),

m\_engine->availableAudioOutputDevices(),

this))

{

m\_spectrograph->setParams(SpectrumNumBands, SpectrumLowFreq, SpectrumHighFreq);

// QCoreApplication::instance()->installEventFilter(this);

createUi();

connectUi();

}

MainWidget::~*MainWidget*()

{

}

//-----------------------------------------------------------------------------

// Public slots

//-----------------------------------------------------------------------------

void MainWidget::stateChanged(QAudio::Mode mode, QAudio::State state)

{

Q\_UNUSED(mode);

if (QAudio::ActiveState != state && QAudio::SuspendedState != state) {

m\_levelMeter->reset();

m\_spectrograph->reset();

}

}

void MainWidget::formatChanged(const QAudioFormat &format)

{

dataSource->setFormat(format);

m\_spectrograph->setFormat(format);

#ifndef DISABLE\_WAVEFORM

if (QAudioFormat() != format) {

m\_waveform->initialize(format, WaveformTileLength,

WaveformWindowDuration);

}

#endif

}

void MainWidget::spectrumChanged(qint64 position, qint64 length,

const FrequencySpectrum &spectrum)

{

m\_progressBar->windowChanged(position, length);

// m\_spectrograph->spectrumChanged(spectrum);

}

void MainWidget::errorMessage(const QString &heading, const QString &detail)

{

QMessageBox::warning(this, heading, detail, QMessageBox::Close);

}

void MainWidget::*timerEvent*(QTimerEvent \*event)

{

}

void MainWidget::audioPositionChanged(qint64 position)

{

#ifndef DISABLE\_WAVEFORM

m\_waveform->audioPositionChanged(position);

#else

Q\_UNUSED(position)

#endif

}

void MainWidget::bufferLengthChanged(qint64 length)

{

m\_progressBar->bufferLengthChanged(length);

}

//-----------------------------------------------------------------------------

// Private slots

//-----------------------------------------------------------------------------

void MainWidget::showFileDialog()

{

const QString dir;

const QStringList fileNames = QFileDialog::getOpenFileNames(this, tr("Open WAV file"), dir, "\*.wav");

if (fileNames.count() && QFile(fileNames.front()).exists()) {

reset();

setMode(LoadFileMode);

m\_engine->loadFile(fileNames.front());

}

m\_engine->startPlayback();

m\_engine->suspend();

}

void MainWidget::aboutQt()

{

QMessageBox::aboutQt(this);

}

void MainWidget::showSettingsDialog()

{

m\_settingsDialog->*exec*();

if (m\_settingsDialog->result() == QDialog::Accepted) {

m\_engine->setAudioInputDevice(m\_settingsDialog->inputDevice());

m\_engine->setAudioOutputDevice(m\_settingsDialog->outputDevice());

m\_engine->setWindowFunction(m\_settingsDialog->windowFunction());

}

/\*

QSize size = viewer.size();

viewer.setFixedSize(60, 60);

viewer.grabFramebuffer().save("test.png");

viewer.setFixedSize(size);

viewer.setAutoFillBackground(true);

\*/

}

void MainWidget::getScreenShot(){

/\*QSize size = viewer.size();

viewer.setFixedSize(9000, ((double)size.height()/size.width())\*9000);

QImage img = viewer.grabFramebuffer();

viewer.setFixedSize(size);

viewer.setAutoFillBackground(true);\*/

QImage img = viewer.grabFramebuffer();

QString filter;

QString path = QFileDialog::getSaveFileName(this, QString(), "screenShot.png","PNG (\*.png);;JPG (\*.jpg);; GIF (\*.gif)", &filter);

if(path.isEmpty() == false)

{

if(filter.isEmpty())

path += ".png";

img.save(path);

}

}

void MainWidget::initializeRecord()

{

reset();

setMode(RecordMode);

if (m\_engine->initializeRecord())

{

m\_engine->startRecording();

///////////////////////////////////////////////////////////RECORD START

}

}

void MainWidget::play\_pause()

{

if(m\_engine->state() == QAudio::SuspendedState || m\_engine->state() == QAudio::StoppedState)

m\_engine->startPlayback();

else

if(m\_engine->state() == QAudio::ActiveState)

m\_engine->suspend();

}

bool MainWidget::isFullScreen()

{

return QMainWindow::isFullScreen();

}

void MainWidget::stop()

{

m\_engine->stopPlayback();

}

bool MainWidget::saveDump()

{

QBuffer buf;

buf.*open*(QBuffer::ReadWrite);

m\_engine->getFilterTempateForPlay()->save(0, m\_engine->getDuration(), &buf);

WavFile::saveWAV("test.wav", buf.data(), m\_engine->format());

}

double MainWidget::getDuration()

{

return m\_engine->getDuration();

}

double MainWidget::getTimePosition()

{

return m\_engine->getTimePosition();

}

void MainWidget::setTimePosition(double time)

{

m\_engine->setTimePosition(time);

}

Graph \*MainWidget::addPraph(QString name, int idView)

{

Graph \*nGraph = new Graph(qrand(), name, 0, Graph::WAVE, getTemplatesQML().first(), this);

graphs[idView].insert(nGraph->getId(), nGraph);

return nGraph;

}

QList<int> MainWidget::getPraphIds(int idView)

{

return graphs[idView].keys();

}

Graph \*MainWidget::getPraphById(int id, int idView)

{

return graphs[idView][id];

}

QStringList MainWidget::getTemplatesQML()

{

return m\_engine->getCreatedTemplates().keys();;

}

bool MainWidget::subscribeToTemplate(Graph \*graph, QXYSeries \*set, int chanel)

{

if(graph == NULL)

return false;

GraphFilterService\* servise = m\_engine->getCreatedTemplates().value(graph->getTemplateName());

if(servise == NULL)

return false;

switch (graph->getType()) {

case Graph::SPECTR:

servise->subscribeSpectrum(set, chanel);

break;

case Graph::WAVE:

servise->subscribeWaveForm(set, chanel);

break;

default:

return false;

break;

}

return true;

}

bool MainWidget::unSubscribeToTemplate(Graph \*graph, QXYSeries \*set, int chanel)

{

if(graph == NULL)

return false;

GraphFilterService\* servise = m\_engine->getCreatedTemplates().value(graph->getTemplateName());

if(servise == NULL)

return false;

switch (graph->getType()) {

case Graph::SPECTR:

servise->unSubscribeSpectrum(set, chanel);

break;

case Graph::WAVE:

servise->unSubscribeWaveFormm(set, chanel);

break;

default:

return false;

break;

}

return true;

}

void MainWidget::changePlayTemplate(QString name)

{

m\_engine->setPlayTemplate(name);

}

void MainWidget::*keyReleaseEvent*(QKeyEvent \*event)

{

QMainWindow::*keyReleaseEvent*(event);

switch (event->key()) {

case Qt::Key\_F11:

switchFullScreen();

break;

default:

break;

}

}

void MainWidget::*keyPressEvent*(QKeyEvent \*event)

{

QMainWindow::*keyPressEvent*(event);

event->setAccepted(false);

}

bool MainWidget::*eventFilter*(QObject \*Object, QEvent \*Event)

{

if (Event->type() == QEvent::KeyPress)

{

QKeyEvent \*keyEvent = (QKeyEvent\*)Event;

*keyReleaseEvent*(keyEvent);

}

}

//-----------------------------------------------------------------------------

// Private functions

//-----------------------------------------------------------------------------

void MainWidget::switchFullScreen()

{

if(this->isFullScreen())

this->showNormal();

else

this->showFullScreen();

}

void MainWidget::createUi()

{

/\*viewer.engine()->addImportPath(extraImportPath.arg(QGuiApplication::applicationDirPath(),

QString::fromLatin1("qml")));\*/

QObject::connect(viewer.engine(), &QQmlEngine::quit, this, &QMainWindow::close);

ui->setupUi(this);

this->setWindowTitle("Spectrum");

// this->setWindowIcon(QIcon(":/audio-spectrum-xxl.png"));

// this->showFullScreen();

dataSource = new WaveFormCustom(this);

viewer.rootContext()->setContextProperty("dataSpectr", m\_spectrograph);

viewer.rootContext()->setContextProperty("dataSource", dataSource);

viewer.rootContext()->setContextProperty("mainWindow", this);

viewer.setSource(QUrl("qrc:/qml/qmloscilloscope/main.qml"));

viewer.setResizeMode(QQuickWidget::SizeRootObjectToView);

//viewer.setColor(QColor("#404040"));

// viewer.show();

ui->gridLayout->addWidget(&viewer);

this->setFocusPolicy(Qt::StrongFocus);

//viewer.setFocusProxy(this);

setFocusProxy(&viewer);

//viewer.setFocusPolicy(Qt::StrongFocus);

viewer.grabKeyboard();

}

void MainWidget::connectUi()

{

CHECKED\_CONNECT(m\_engine, SIGNAL(stateChanged(QAudio::Mode,QAudio::State)),

this, SLOT(stateChanged(QAudio::Mode,QAudio::State)));

CHECKED\_CONNECT(m\_engine, SIGNAL(formatChanged(const QAudioFormat &)),

this, SLOT(formatChanged(const QAudioFormat &)));

m\_progressBar->bufferLengthChanged(m\_engine->bufferLength());

CHECKED\_CONNECT(m\_engine, SIGNAL(bufferLengthChanged(qint64)),

this, SLOT(bufferLengthChanged(qint64)));

CHECKED\_CONNECT(m\_engine, SIGNAL(recordPositionChanged(qint64)),

m\_progressBar, SLOT(recordPositionChanged(qint64)));

CHECKED\_CONNECT(m\_engine, SIGNAL(playPositionChanged(qint64)),

m\_progressBar, SLOT(playPositionChanged(qint64)));

CHECKED\_CONNECT(m\_engine, SIGNAL(recordPositionChanged(qint64)),

this, SLOT(audioPositionChanged(qint64)));

CHECKED\_CONNECT(m\_engine, SIGNAL(playPositionChanged(qint64)),

this, SLOT(audioPositionChanged(qint64)));

CHECKED\_CONNECT(m\_engine, SIGNAL(levelChanged(qreal, qreal, int)),

m\_levelMeter, SLOT(levelChanged(qreal, qreal, int)));

CHECKED\_CONNECT(m\_engine, SIGNAL(spectrumChanged(qint64, qint64, const FrequencySpectrum &)),

this, SLOT(spectrumChanged(qint64, qint64, const FrequencySpectrum &)));

}

void MainWidget::reset()

{

#ifndef DISABLE\_WAVEFORM

m\_waveform->reset();

#endif

m\_engine->reset();

m\_levelMeter->reset();

m\_spectrograph->reset();

m\_progressBar->reset();

}

void MainWidget::setMode(Mode mode)

{

m\_mode = mode;

emit modeChanged(mode);

}

ДОДАТОК Б

Код головного вікна QML

import QtQuick 2.0

//![1]

Rectangle {

id: *root*

width: 600

height: 400

color: *root*.screenShot ? "#fff" : "#2c2d37"

property int nextId: 0

property bool screenShot: false

Row{

id: *windowInfoPanel*

anchors.top: *parent*.top

anchors.right: *parent*.right

anchors.left: *parent*.left

height: 30

spacing: 5

IconBtn{

id: *aboutQt*

url: "qrc:/icons/qml/icons/ic\_help\_white\_24px.svg"

onClick: {

*mainWindow*.aboutQt();

}

}

}

Row{

id: *windowControlPanel*

anchors.top: *parent*.top

anchors.right: *parent*.right

height: 30

spacing: 5

IconBtn{

id: *screenShot*

url: "qrc:/icons/qml/icons/screen-1.svg"

onClick: {

*root*.screenShot = true;

function Timer() {

return *Qt*.createQmlObject( "import QtQuick 2.0; Timer {}", *root*);

}

var *timer* = new Timer();

*timer*.interval = 1000;

*timer*.repeat = false;

*timer*.triggered.connect(function () {

*mainWindow*.getScreenShot();

*root*.screenShot = false;

})

*timer*.start();

}

}

IconBtn{

id: *fullScreenSwitch*

url: "qrc:/icons/qml/icons/ic\_open\_with\_white\_48px.svg"

onClick: {

*mainWindow*.switchFullScreen();

*closeBtn*.visible = *mainWindow*.isFullScreen();

}

}

IconBtn{

id: *closeBtn*

url: "qrc:/icons/qml/icons/ic\_power\_settings\_new\_white\_24px.svg"

visible: *mainWindow*.isFullScreen();

onClick: *mainWindow*.close();

Connections{

target: *root*

onWidthChanged:{

*closeBtn*.visible = *mainWindow*.isFullScreen();;

}

}

}

}

TabView{

width: *parent*.width

anchors.fill: *parent*

anchors.margins: 20

anchors.bottomMargin: *mediaControlPanel*.height + 3

templateName: "Page"

}

MediaControlPanel{

id: *mediaControlPanel*

z:2//for MediaControlPanel overlap

anchors.bottom: *parent*.bottom

}

}