Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмна система для налаштування гітар з використанням засобів Windows Forms
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДППЗ.180123.01.15.ПЗ

Виконав студент IV курсу група ІПЗ-18-1 _____ Підпис Д. І. Храмцов Ініціали, прізвище

Керівник канд. техн. наук, доцент Науковий ступінь, звання _____ Підпис І. В. Гурман Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Підпис Ю. В. Форкун Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Підпис Л. П. Бедратюк Ініціали, прізвище

6 червня 2021 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
5.02 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Храмцову Данилу Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмна система для налаштування гітар з використанням засобів Windows Forms

Керівник проекту (роботи) Гурман Іван Васильович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2022 р. № 18
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2022 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 10 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання пр
Нормоконтроль	Форкун Ю.В., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 05 » лютого 2022р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Прим
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	Травень 2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	**з 01.06.2022**	

Студент _____ Підпис Д. І. Храмцов Ініціали, прізвище

Керівник проекту (роботи) _____ Підпис І. В. Гурман Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Програмна система для налаштування гітар з використанням засобів Windows Forms.

Автор проекту: Храмцов Данило Ігорович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 85 с., 19 рис., 10 табл., 13 джерел.

Графічна частина: 10 слайдів.

ГІТАРНИЙ ТЮНЕР, ТЮНЕР, ДОДАТОК ДЛЯ КОМП'ЮТЕРА, WINDOWS FORMS.

Метою цієї дипломної розробки є програмної системи для налаштування гітар з використання засобів Windows Forms, для вирішення таких проблем як безкоштовний функціонал та зручного функціоналу.

У моєму дипломному проекті було проаналізовано предметну область, визначено функціональні задачі та розроблено ТЗ. Проведено аналіз всіх інструментів та технологій розробки на Windows Forms і бібліотек для роботи зі звуком та спроектовано архітектуру разом з функціональним інтерфейсом. На основі цієї інформації було розроблено додаток для налаштування гітари. Після розробки програмний продукт пройшов функціональне, модульне та конфігураційне тестування.

Для розробки додатку була використана мова програмування C#, фреймворк Windows Forms. Програмний продукт дозволяє швидко та зручно налаштувати свій музичний інструмент.

03.06.2022
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПШПЗ.180123.01.15.ПЗ	Пояснювальна записка	60		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	10		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПШПЗ.180123.01.15.ВД		
Виконав		Храмцов Д.І.		3.06	Програмна система для налаштування гітар з використанням засобів Windows Forms	Літ.	Арк. 1 / Акрушів 1
Керівник		Гурман І.В.		3.06			
Н. Контр.		Форкун Ю.В.		7.06	Відомість документів	ХНУ, ІПЗ-18-1	
Зав. Каф.		Бедратюк Л.П.		6.06			

ЗМІСТ

ВСТУП

На даний час немає таких людей для яких би було таємницею те, що гітара повинна бути налаштована, для гарного та динамічного звучання музичного інструмента, але у початківця можуть виникнути серйозні перешкоди, граючи на інструменті неправильно або неточно налаштованому, ви не тільки шкодите своєму музичному слуху та й гальмуєте його розвиток, для вирішення цієї проблеми нам допоможе тюнер для налаштування гітари.

В даний час існує безліч тюнерів для налаштування вашого інструменту, наприклад встроєний тюнер в саму гітару, так звані пружинні тюнери, віртуальні тюнери, педальні тюнери та багато інших. Але у кожного є свої недоліки, за якісь потрібно відати велику суму, деякі не завжди є під поряд, а більшість працює не зовсім коректно. В якості альтернативи вирішення проблеми буде використовуватись спеціалізована програма яка завжди буде під рукою.

Актуальність теми дипломного проекту обумовленна у необхідністю створення простого програмного продукту, у якому повний функціонал без дратівливої вбудованої реклами яка повноцінно заміняє фізичний тюнер.

Метою мого дипломного проектування та переддипломної практики є розробка програмної системи для налаштування гітар з використання засобів Windows Forms . Надалі для досягнення поставленої мети були сформовані завдання на дипломне проектування:

– дослідити предметну область та розкрити потреби користувачів програмного продукту;

– розробити найбільш зручний і зрозумілий інтерфейс користувача;

– переглянути додатки які існують в цій галузі та сформувати вимоги на розробку нового програмного забезпечення відповідно до потреб часу;

– проаналізувати алгоритм частоти струн та інтегруючи їх у власну систему;

– виконати програмну реалізацію проекту;

– проводити тести та випробування програмного продукту;

Розробка цього програмного забезпечення буде корисна не тільки для музикантів-початківців, але й професіоналам які приділяють величезну увагу точному налаштуванню інструменту за допомогою тюнера, що найчастіше пов'язано зі специфічними умовами, в яких музикант не має можливості налаштуватися по слуху.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Windows Forms – це платформа для створення додатків Windows. Він надає один із найефективніших способів створення класичних програм за допомогою візуального конструктора у Visual Studio. Представляє такі функції, як розміщення візуальних елементів керування шляхом перетягування, полегшують створення класичних додатків. Також необхідно зауважити ще про такі технології як: WPF, UWP. Однак більш простішою платформою є Windows Forms або скорочено WinForms.

Об’єктом розробки є програмна система для налаштування гітари, яка дозволяє дізнатися про правильну частоту струн і знайти ноти. Здійснюється це за допомогою наступних допоміжних бібліотек:

1. NAudio це аудіобібліотека. NET з відкритим вихідним кодом, написана Марком Хітом, за допомогою цієї бібліотеки стало можливим відтворення аудіо за допомогою різних API, читання аудіо з багатьох стандартних форматів файлів, перетворення між різними формами несжатого аудіо та багато інших корисних функцій;

2. Accord.NET Framework – є фреймворк машинного навчання .NET, який підкорює більшість аудіо- та відеобібліотек, написаних на C#. Основна платформа для створення програмного забезпечення для комп'ютерного моніторингу, комп'ютерної обробки, обробки сигналів та статистики на промисловому рівні.

3. AForge.NET – це середовище C#, яке отримало визнання за дослідження в галузі інформатики та інтелектуальної обробки даних: обробка зображень, нейронні мережі, генетичні алгоритми, машинне навчання, робототехніка та інше.

4. Simple Injector – це проста у використанні бібліотека впровадження залежностей (DI) для .NET 4.5, .NET Core, .NET 5, .NET Standard, UWP, Mono та Xamarin. Simple Injector легко інтегрується з такими платформами як Web API, MVC, WCF, ASP.NET Core і багатьма іншими. Simple Injector має ретельно відібраний набір функцій у своїй основній бібліотеці для підтримки багатьох розширених сценаріїв.

Отже було проведено певний аналіз предметної області, визначено суть структурних та функціональних особливостей. Визначено та частково описаний об’єкт розробки.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

В даний час існує безліч популярних онлайн-майданчиків для публікації доповнень на платформі Windows, ви можете скористатися такою платформою, як Microsoft Store, доповненням від Microsoft. Зробивши огляд ринку можна побачити просування доповнень, які завоювали популярність серед користувачів:

– Fine Chromatic Tuner;

– GuitarTuna;

– AP GUITAR TUNER;

Було проведено аналіз усіх перерахованих додатків з урахуванням їхньої вартості, функціональних характеристик, надмірностей і недоліків, з урахуванням відгуків користувачів та особистого досвіду використання.

У додатка Guitar Tuna на сторінці в softonic автор заявиві наступні функціональні особливості:

– можливість налаштування гітари, чи то акустичної, чи електронної;

– додаток містить камертон, він може генерувати тони від 1 до 22050 Гц.

– є вбудований метроном;

– є можливість отримати доступ до повної бібліотеки акордів, де міститься багато різних схем акордів.

Для цієї програми, яка доступна для доступний комп’ютерів з версією операційної системи Windows 7 та новіших версій. Продукт ментально безкоштовний, а отже за завантажити додаток можна безкоштовно, але щоб скористатись його функціоналом потрібно купувати підписку, також без неї в додатку э реклама яка заважає, забирає час та відволікає. (рисунок 2.1).

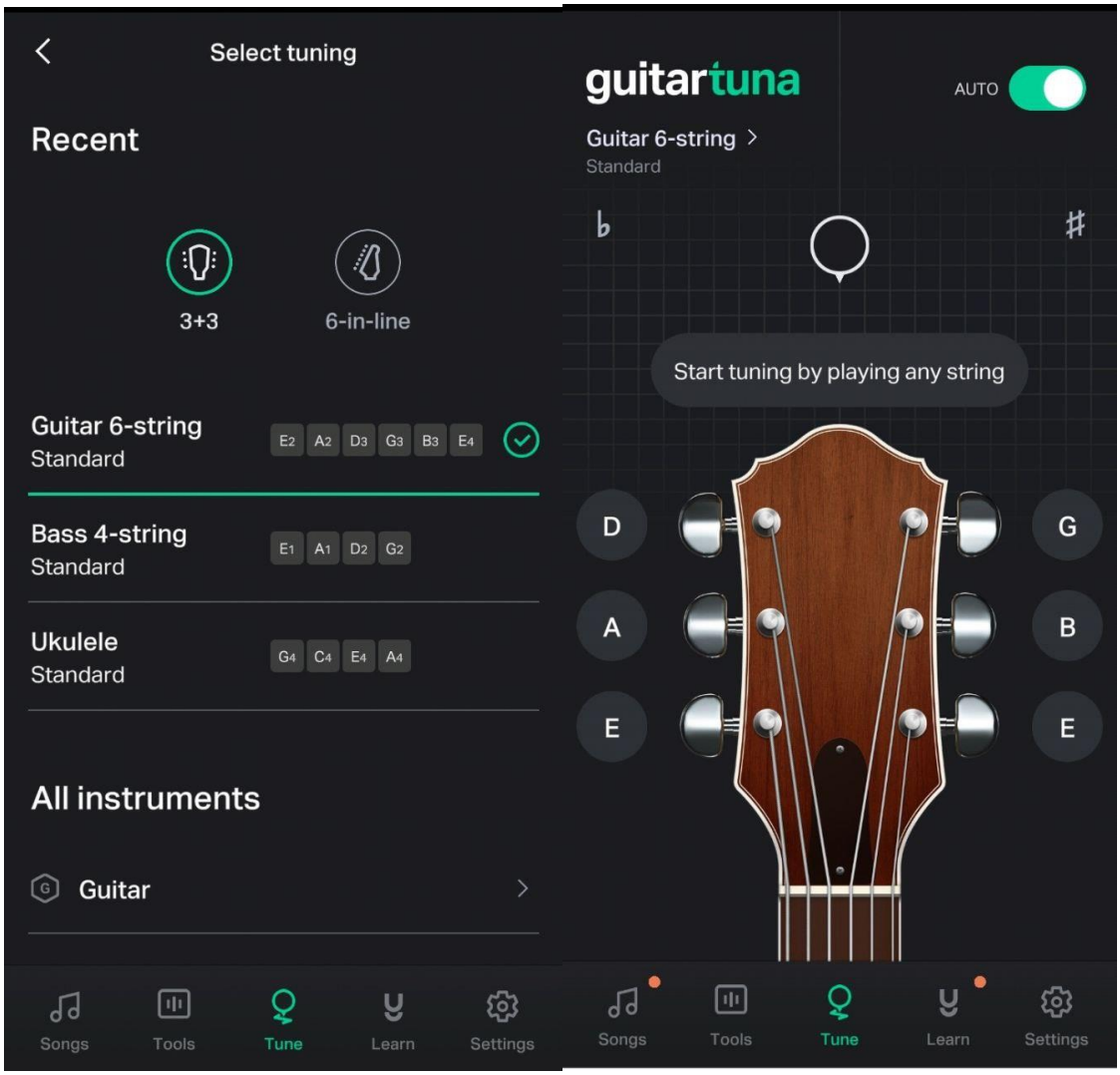


Рисунок 1.1 – інтерфейс додатку Guitar Tuna

Далі йде програма під назвою AP Guitar tuner. Розробник акцентує увагу на те що додаток безкоштовний У новому є лише необхідні функції та простіший інтерфейс. Утиліта проста та стабільна. Сам тюнер допоможе вам налаштувати гітару за допомогою чудового мікрофона, підключеного до комп'ютера. З поміж всіх функціональних можливостей програми розробник виділяє наступні:

– простий інтерфейс;

– зрозумілий функціонал;

– безкоштовна для використовування.

Цей програмний продукт також доступний для операційної системи Windows 7 та пізніших версій. На передньому вигляді інтерфейс (рисунок 2.2) дуже мінімалістичний. У цій версії додатку реклама відсутня.

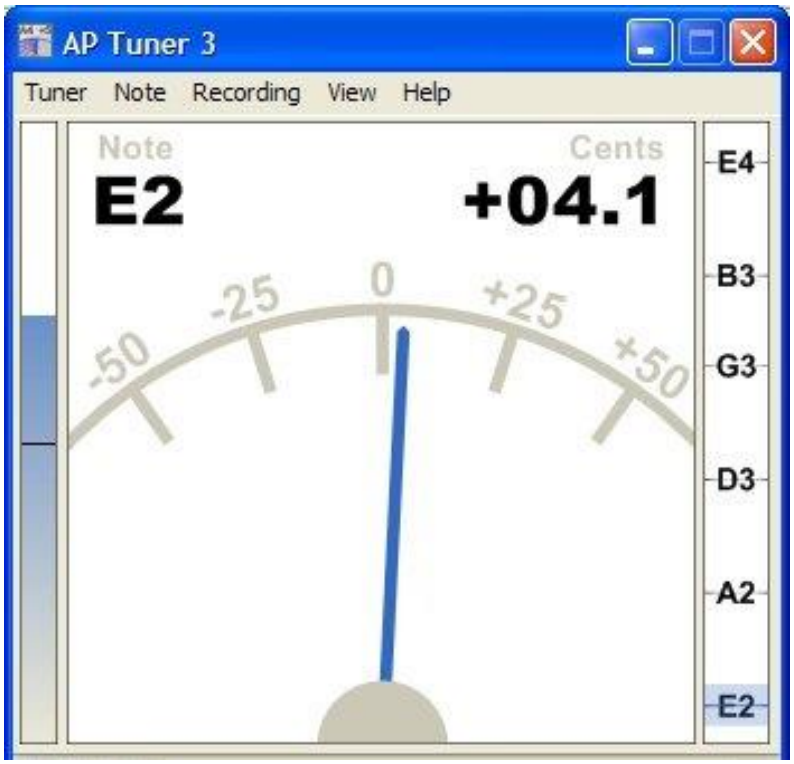


Рисунок 1.2 – Інтерфейс додатку AP Guitar tuner

Недостатками є відсутність достатнього програмного забезпечення, навіть мінімальної функціональності програми, навіть частина функціональності недопрацьована чи працює.

Далі йде хроматичний тюнер Fine Chromatic Tuner. Дилер описує його доповнення: Fine Chromatic Tuner - професійний хроматичний тюнер з унікальними алгоритмами, розбитими на частини для найточнішого налаштування. Вона добре підходить для широкого спектру інструментів і шанується з гітарою, скрипкою, флейтою, гавайськими гральними кістками, кількома дерев'яними духовими інструментами та мідними духовими

інструментами з тонким голосом.. Також автор виділив наступні особливості цього застосунку:

– робочий діапазон програми становить від 70 Гц до 550 Гц;

– зрозумілий функціонал;

– є безкоштовним для використовування.

Додаток (рисунок 2.3) написаний у спокійних колірних гаммах, він цілком прийнятний для ока. Недоліком можна відмітити що протягом часу не вдалося відобразити правильну анімацію самої стрілки.

Програмний продукт з назвою Fine Chromatic Tuner доступний для Wsndows 7 і пізніших версій і безкоштовний. Також плюсом що у додатку відсутня реклама.

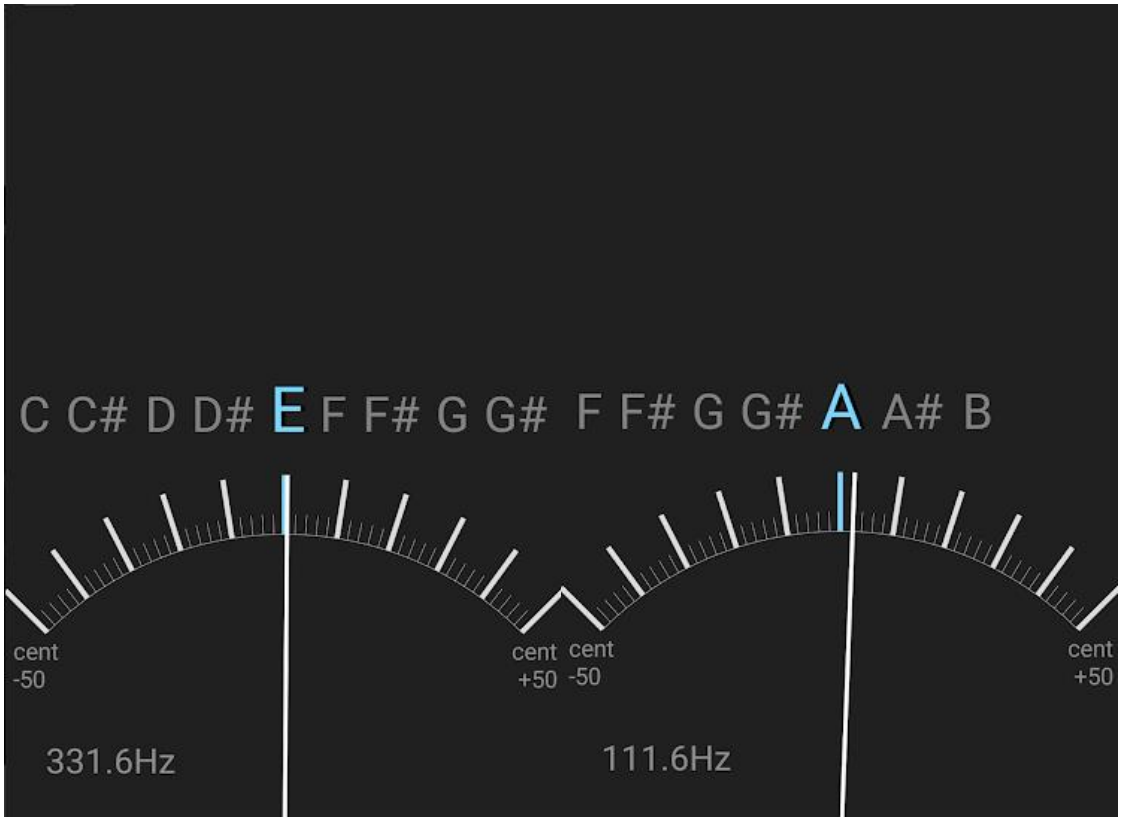


Рисунок 1.3 – Інтерфейс додатку Fine Chromatic Tuner

Отже, вивчивши ринок додатків для налаштування гітари, можна зробити висновок, що на даний момент переважна більшість програм на ринку мають хорошу функціональність, але більшість із них відносно безкоштовні та містять

багато реклами, або навпаки, програма повністю безкоштовна, але має дуже мало функціональності, щоб гарантувати якість програми.

Для більш детального представлення та порівняння гітарних тюнерів (з точки зору їх використання як додатку для настроювання гітари) вони були проаналізовані за деякими ключовими критеріями. На основі цих даних було складено порівняльну таблицю наявного програмного забезпечення (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика наявного ПЗ

Назва	Наявність реклами в додатку	Відображення частоти	Відображення нот	Коректна робота додатка	Платний функціонал
Fine Chromatic Tuner	Ні	Так	Так	Ні	Ні
AP Guitar tuner	Ні	Ні	Так	Так	Ні
Guitar Tuna	Так	Так	Так	Так	Так

Проведений аналіз подібного програмного забезпечення показав, що розроблюваний додаток повинен:

– Мати безкоштовний функціонал без встроєнної реклами в додатку та безкоштовну робочу частину;

– повинно відображати для користувача такі данні як частоту та ноти для більш кращого функціоналу;

– Всі функціональні графічні та нефункціональні елементи додатку повинні працювати коректно.

Отже після проведеного аналізу та вивчення ринку програмних систем для налаштування гітари, можна зробити висновок що на даний момент переважна частина тюнерів, які представленні на ринку мають непогане функціональне наповнення але є частково платними чи мають спливаючу рекламу, або є

повністю безкоштовними, але не мають відображення важливої інформації, а деякі взагалі некоректно працюють.

1.3 Визначення вимог до програмної системи

Після аналізу предметної області та популярних додатків які зараз представлені на ринку було вирішено розробити новий додаток. На відміну від наявних програм про які йшла мова, мій додаток створювався з метою надання зручного інструменту для налаштування гітар.

Враховуючи аналіз предметної області сформуємо перелік функціональних можливостей, які повинна надавати програма, та покажемо що саме користувач зможе зробити за допомогою діаграми варіантів використання (ВВ).

Функціональні можливості:

– Перегляд частоти звуку;

– можливість визначати ноти звука;

– можливість бачити правильність частоти струн.

Вимоги до інтерфейсу:

– Правильна робота візуальних частин;

– сучасний вигляд інтерфейсу;

– На головному екрані повинно бути візуальна частина яка показує правильність настроювання частоти струн;

– Спеціальне елемент для відображення частоти;

– Відображення нот.

Програмна система може використовуватися на будь-якому пристрої з операційною системою Windows 7 та вище, попередньо встановивши цей додаток, додаткових налаштувань додаток не потребує, для початку роботи з ним.

Сформулюємо таблицю у якій подано опис варіантів використання (таблиця 1.2).

Таблиця 1.2 – Опис варіантів використання

Актор	Назва ВВ	Опис ВВ
1	2	3
Користувач	Вибрати мікрофон для запису звука	Користувач може сам вибрати як зчитувати звук і з якого мікрофона, який повинен бути з’єднаний з пристроєм та встановленим на ньому додатком.
	Вибрати ноту або струну яку потрібно налаштувати	Для комфорту налаштування дозволяє вибрати ноту яку потрібно налаштувати для певної струни.
	Налаштування на іншу тональність	Дозволяє перелаштувати акустичну гітару на тональність вище або нижче за стандартний стрій.
	Налаштування на іншу ноту	Тим хто цього потребою також можливо настроювати чи просто визначати частоту струн.

Відповідно до описаних акторів та варіантів використання було побудовано діаграму, яка зображена на рисунку 1.4 та подано у додатку Б.

Саме технічне завдання на розробку програмної системи для налаштування гітари подано у додатку А.

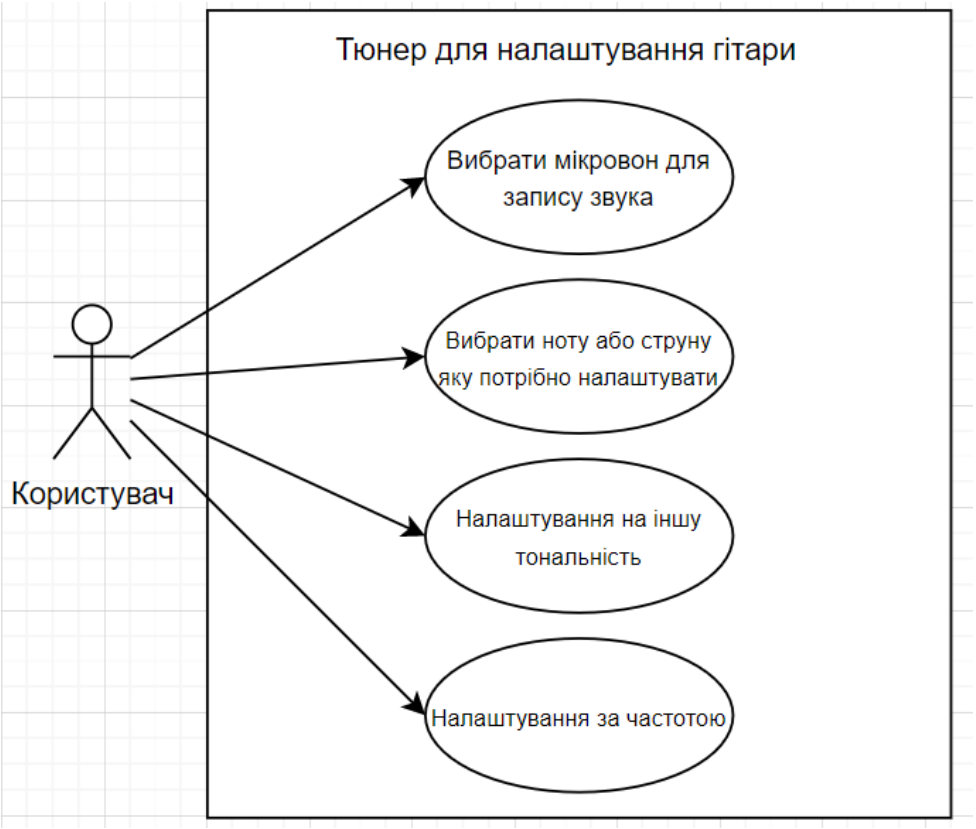


Рисунок 1.4 – Діаграма варіантів використання

Отже, після проведення аналізу предметної області було розглянуто програмне забезпечення для налаштування гітари. Відповідно було визначено її переваги та недоліки, а також необхідність розробки саме такого програмного продукту. Вимоги до програмної системи було за допомогою діаграми використання.

2 ПРОЕКТУВАННЯ ДОДАТКА

2.1 Архітектура та функціональна структура додатка

В розробці програмного забезпечення одним з найважливіших етапів є проектування, оскільки добре продумана архітектура програми дозволяє легко розробляти та в подальшому підтримувати розроблений програмний продукт. Проте якщо взяти архітектуро одного вдалого продукту та застосувати до іншого проекту результат може бути жахливим. Це відбувається із–за того що архітектура однозначно не буває хорошою чи поганою її оцінка є дуже суб'єктивним процесом, і сильно залежить від тих задач які були поставленні.

Архітектура системи - це фундаментальні концепції чи властивості системи у своєму оточенні, втіленому в його елементах, відносинах та принципах. Також її називають структурою програми чи комп'ютерної системи, куди входять програмні компоненти, зовні видимі властивості цих компонентів та відносини з-поміж них. Цей термін також застосовується до документації з архітектури програмного забезпечення.

Архітектурна система яку ми будемо розбирати називається Apple MVC, яка трохи краща за стандартну Classic MVC. Шаблон зображено на рисунку 2.1.

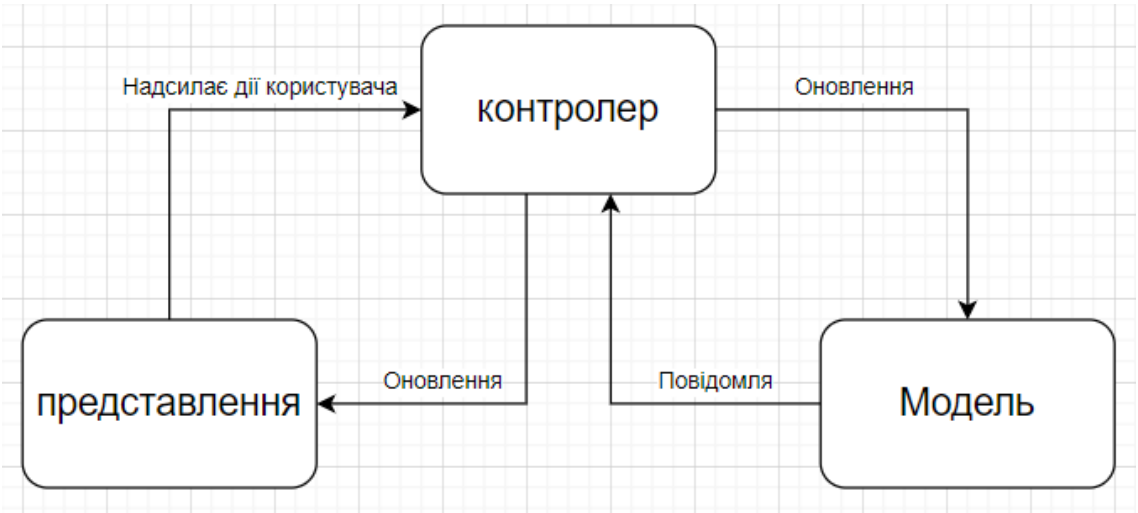


Рисунок 2.1 – Схема шаблону MVC

Кожна буква абревіатури має таке значення:

M – Model. У ролі моделі можуть виступати як прості класи чи структури, і складні сутності, що вимагають інформації з бази даних або мережі. Модель не знає про існування інтерфейсу користувача (UI). Його можна розглядати як ядро програми, яке можна використовувати повторно з іншим інтерфейсом. Наприклад, використовувати ту саму модель для версії програми для різних пристроїв

V – View. В'ю – це уявлення. Те, що користувач бачить та з чим взаємодіє. Відповідальності в'ю – це відображення себе на екрані та реагування на дії користувача, але воно не містить логіки та просто передає подію у контролеру. Як уявлення в C#-додатках використовуються Форма та його спадкоємці. Одну і ту ж в'ю можна використовувати на різних екранах програми або навіть перенести до іншого.

C – Controller. Це сполучна ланка між моделлю та уявленням. У його обов'язки входить прийом дій користувача з уявлення, їх обробка та, при необхідності, оновлення моделі і оновлення уявлення при зміні моделі. Оскільки контролери містять логіку, специфічну для конкретної програми, їх важко використати повторно.

Ця модель також називається пасивним уявленням. Існує також різновид цієї моделі під назвою Massive View Controller. Основна відмінність полягає в тому, що активність (екран програми) разом з його логікою має розглядатися як шар уявлення, а для контролера має бути створений окремий клас.

Архітектура MVC є дуже гарним та практичним вибором для даного програмного забезпечення.

Очевидним розвитком MVC є MVP. Оскільки ViewController є частиною вистави, замість нього нам потрібно додати якусь іншу сутність, що не має відношення до вистави, тому ми отримуємо ведучого. Схема яка показана на рисунку 2.2 дуже схожа на MVC від Apple. Різниця полягає тільки в тому щ вся логіка не зв’язана з представленням, переходить із ViewController в Presenter.

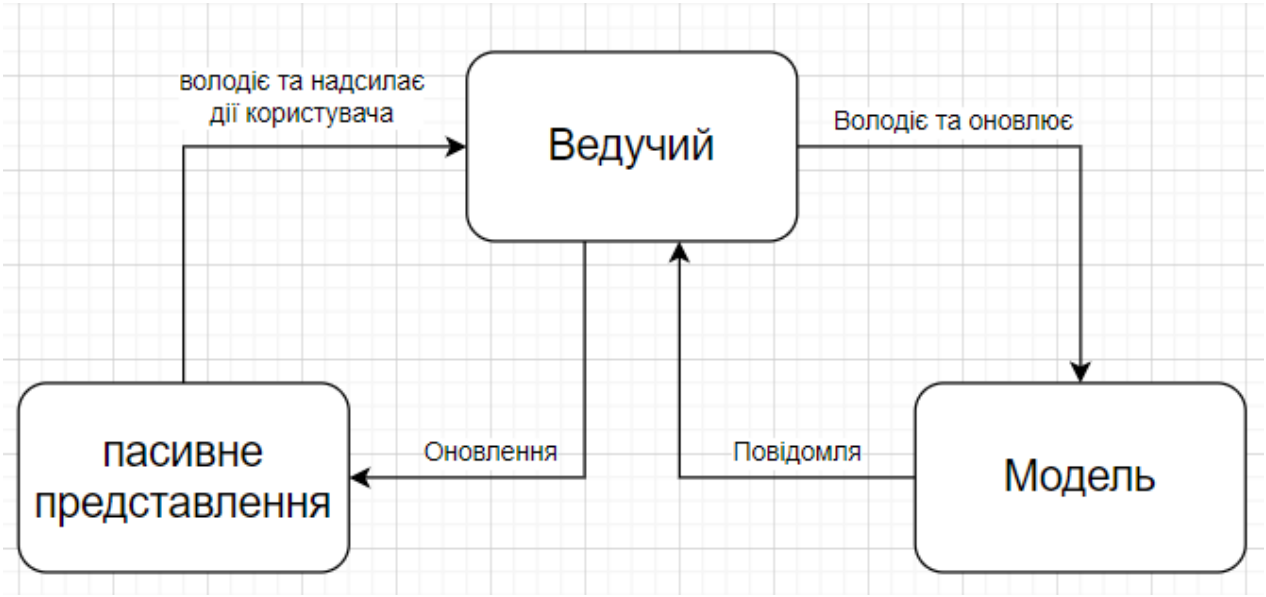


Рисунок 2.2 – Схема шаблону MVP

Представник є багаторазовим і може бути замінений залежно від платформи, де працює додаток, а уявлення можуть бути реалізовані декількома інтерфейсами. Це робить цю модель набагато більш гнучкою, ніж MVC, та спрощує супровід та розширення існуючого коду. Однак за такого підходу вам доведеться написати багато коду як у поданні, так і в представнику через необхідність створення та реалізації всіх інтерфейсів. Оскільки кожен інтерфейс повинен охоплювати лише невелику частину взаємозалежних взаємодій, у результаті таких інтерфейсів може бути багато.

MVVM дуже схожий на MVP. Різниця між ними полягає в тому, що замість презентатора використовується ViewModel, який зберігає стан представлення. Наприклад, текст який зображено на екрані або значення поля введення. Але на відміну від MVP, уявлення саме приймає дані для відображення. Це можна зробити у різний спосіб, але часто використовується зв'язування даних (схема в наступному розділі). Це підхід, у якому один об'єкт слідкує за станом іншого. Ті. представлення буде відстежувати значення, що зберігається в ViewModel, і змінювати відображення при його зміні. Шаблон MVVM зображено на рисунку 2.3

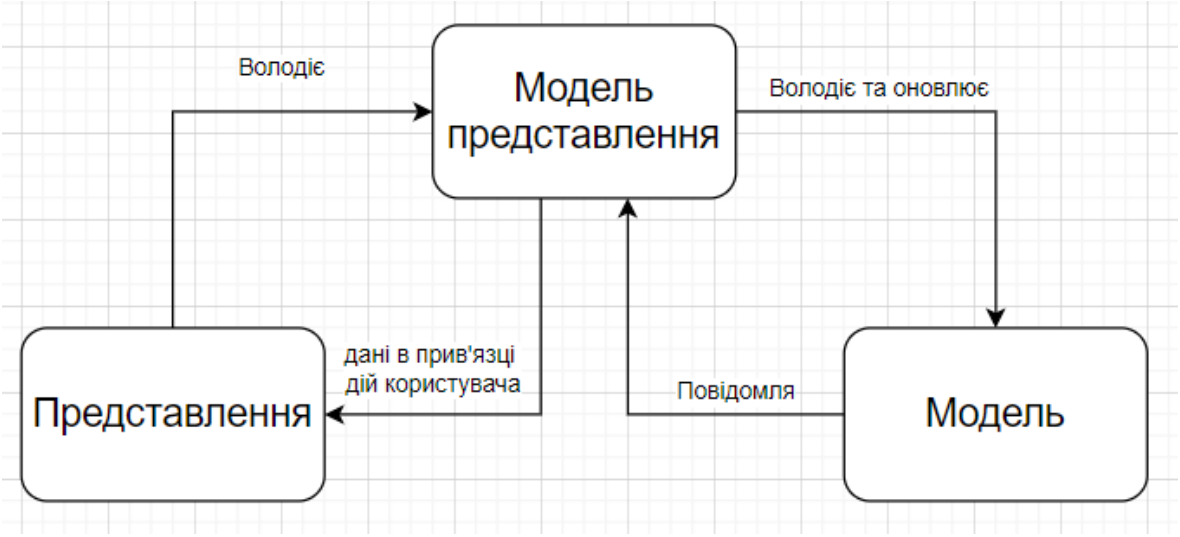


Рисунок 2.3 – Схема шаблону MVVM

MVVM та MVP вирішують проблему великого контролера та спрощують тестування. Однак поріг входу для проектів, написаних на них, дещо вищий. Це особливо актуально для MVVM з прив'язками, оскільки часто у таких проектах прив'язка даних використовується не лише між View та ViewModel.

Причини використання MVVM також включають загальну продуктивність програм, що використовують цю модель. Синтетичні тести показують, що програми, засновані на архітектурі MVVM, у середньому працюють трохи швидше, ніж аналогічні програми, створені за моделлю MVC або MVP.

У цьому проекті можна використовувати ці або інші архітектурні рішення. Особисто я буду використовувати MVVM, оскільки це спростить реалізацію налаштувань та вкладок.

2.2 Аналіз ти вибір технологій для реалізації програмної системи

Існує безліч мов програмування. Одну з них може потребувати корпоративна система, інша необхідна для розробки сайту, третя підійде для вирішення певних завдань із застосунками для Android та iOS. Вибір самої мови програмування є важливою частиною проектування програмного забезпечення.

Вибраною платформою представлено .Net Framework. Microsoft .NET – це є технологія, яка впроваджена компанією Microsoft як платформа також для створення спільних програм для персональних комп'ютерів веб-програми або порталів. Однією з ідей полягає у створенні сумісності серверів, написаних на різних мовах програмування. Цю функцію корпорація Майкрософт відображають як перевагу .NET.

Платформа .NET Framework – це технологія, яка підтримує створення та виконання багатьох додатків та веб-служб. При розробці .NET Framework основними завданнями були:

– програмне середовище, яке повинно мінімізувати конфлікти при поширенні програмного забезпечення та керуванні версіями;

– забезпечення середовища продуктивності, що повинно забезпечувати безпечне виконання коду програмного додатка та включаючи код, створений невідомим або ненадійним виробником третьої сторони;

– забезпечення середовища виконання коду, що повинно виключати проблеми з виконанням середовищ яка виконує сценарії або інтерпретацією коду;

– забезпечення уніфікованих принципів розробки для самих різних типів програм, таких як програми Windows.

Платформа .NET Framework складається з загальних середовищ для виконання (середовища CLR) та бібліотек класів .NET Framework. Основою платформи .NET Framework є середовище CLR. Середовище виконання може розглядатися як агент, який може керувати кодом під час виконання програмної системи і надавати основні послуги, наприклад як управління пам'яттю чи управління асинхронними потоками і віддалену взаємодію. У такому випадку створюються умови так званої суворої типізації та інших видів контролю правильності виконання коду, які забезпечують безпеку та надійність при роботі. Фактично головним завданням середовища є управління кодом додатка. Код, який звертається до середовища виконання, називається керованим кодом, а код, який не має доступу до середовища, має назву некерований код.

Також платформа може розміщуватися на некерованих компонентах, які підключають середовище CLR до власних процесів і виконують виконання керованого коду, створюючи тим самим програмну оболонку, яка дозволяє використовувати як керований, так і некерований код.

Сама платформа .Net є найкращим фреймворком що зробила компанія. Вона представляє міцну платформу для створення додатків. Взагалі можна виділити наступні частини:

– Підтримка кількох мов. В основі платформи лежить Common Language Runtime (CLR), що означає, що .NET підтримує безліч мов: на додаток до C#, він також включає VB.NET, C++, F# та різні діалекти інших. NET.

– Мультиплатформа. .NET – це є платформа, що може переноситись. Наприклад, остання зараз версія платформи: .NET 6 підтримується на більшості сучасних Windows, MacOS, Linux. Використовуючи різні технології на платформі .NET, ви можете розробляти програми на C# для різних платформ.

– Потужна бібліотека класів. .NET надає єдину бібліотеку класів для всіх мов яка вона підтримує. І незалежно від того, яку програму ми збираємося написати на C # – текстовий редактор, чат або складний веб-сайт — ми так чи інакше будемо використовувати бібліотеку класів .NET.

– Різноманітність технологій. Common Language Runtime (CLR) та базова бібліотека класів є основою для цілого стеку технологій, які розробники можуть використовувати для створення програм.

– Продуктивність. У серії тестів веб-програми .NET 6 значно перевершували веб-програми, створені з використанням інших технологій, у декількох категоріях. Програми на .NET 6 зазвичай характеризуються високою продуктивністю.

Для того щоб реалізувати дану програмну систему було обрано мову програмування C#, на цій мові буде розроблено не тільки весь функціонал додатка та й використовуватись бібліотеки розроблені на цій мові.

На сьогоднішній день язик програмування C# є одним з найпотужніших мов, що швидко розвиваються і затребуваних в IT–секторі. В даний час на ньому

написані найрізноманітніші програми: від невеликих настільних програм до великих веб-порталів та веб-сервісів. C# є об'єктно-орієнтованим і в цьому відношенні багато взяв від Java та C++. Наприклад, C# підтримує поліморфізм, успадкування, перевантаження операторів та статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішувати проблеми побудови великих, але гнучких, масштабованих та розширюваних додатків. А C# продовжує активно розвиватися і з кожною новою версією з'являється все більше цікавих можливостей.

Для того щоб обрати технологію потрібно виділити наступні критерії як злагоджену роботу на вибраній мові та підтримку цієї технології.

Так як в додатку якому створюється є графічна частина, найкраще всього підходить технологія – Window Forms, або скорочено WinForms. Вона найбільше підходить тому що є простою і зручною платформою для використання яка працює лише на операційних системах Windows.

Windows Forms – інтерфейс програмування додатків (API), він відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Задача даного інтерфейсу спрощувати доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді програми. Причому керований код – тобто класи, що реалізують API для Windows Forms, що не залежать від мови розробки. Це означає що програміст однаково може використовувати Windows Forms як при написанні ПЗ на C #, C ++, так і на VB.Net, J # і ін. До переваг технології можна віднести:

- Споживання малої кількості пам’яті;
- розробка всієї логіки виконується на мові програмування C#;
- усі версії операційної системи Windows підтримують цю технологію;
- До недоліків можна віднести:
- Працювати додаток може лише на платформі Windows.
- Набагато менше нових функцій.

Мова програмування C# досить багата на велику кількість технологій та бібліотек, тому щоб реалізувати програмну систему для налаштування гітар, будуть використовуватись наступні бібліотеки:

1. NAudio – це аудіо бібліотека .NET з відкритим вихідним кодом, написана Марком Хітом. За допомогою цієї бібліотеки ви можете відтворювати аудіо, використовуючи різні API, читати аудіо з багатьох стандартних форматів файлів, конвертувати між різними формами несжатого аудіо та багато інших корисних функцій;

2. Accord.NET Framework – це той самий фреймворк машинного навчання .NET, який використовується більшістю аудіо – та відеобібліотек, написаних мовою програмування C#. Це основна платформа для створення програмного забезпечення для моніторингу, комп'ютерного підслуховування, обробки сигналів та статистики на промисловому рівні.

3. AForge.NET – це середовище C#, яке отримало нагороди за дослідження та розробки в галузі комп'ютерних наук та видобутку даних: обробка зображень, нейронні мережі, генетичні алгоритми, машинне навчання, робототехніка та багато іншого.

4. Simple Injector – це проста у використанні бібліотека застосування залежностей (DI) для .NET 4.5, .NET Core, .NET 5, .NET Standard, UWP, Mono і Xamarin. Simple Injector легко інтегрується з такими платформами як Web API, MVC, WCF, ASP.NET Core і багатьма іншими. Simple Injector має ретельно підібраний набір функцій у своїй основній бібліотеці для підтримки багатьох просунутих сценаріїв.

2.3 Аналіз алгоритму роботи програмної системи

Перед початком аналізу самого алгоритму роботи, потрібно зауважити що взаємодія програмної системи та музичного інструмента відбувається через вбудований або під’єднаний зовні мікрофон. Після того додаток може розпізнати пристрій та записувати живий звук або музику. Мікрофон який під’єднано до

звукової карти може приймати цифрові сигнали, які являються сукупністю квантових звукових значень, прийнятих через регулярні проміжки часу. Цифровий сигнал не надає ніякої інформації про частоти, присутні в звукі. Щоб це визначити, необхідно проаналізувати дані.

Короткочасне перетворення Фур'є (STFT) забезпечує представлення фази та амплітуди сигналу. Результат STFT може бути використаний для отримання спектрограми сигналу: квадрат амплітуди за часом та частотами. В даній програмній системі буде використовуватись швидке перетворення Фур'є (FFT) для створення спектрограм сигналу за короткі періоди часу.

Швидке перетворення Фур’є (FFT) – це є алгоритм, що обчислює дискретні перетворення Фур’є (DFT) послідовності – дискретне перетворення Фур’є є інструментом для перетворення певних типів послідовностей функцій в інші типи представлень. Інший спосіб пояснити дискретне перетворення Фур’є полягає в тому, що воно перетворює циклову структуру сигналу в синусоїдальні компоненти. Э три основних алгоритму перетворення Фур’є:

– Дискретне перетворення Фур’є;
– пряме дискретне перетворення Фур’є;
– зворотне дискретне перетворення Фур'є;

В нашому проекті основним алгоритмом буде дискретне перетворення Фур’є виглядає наступним чином, за $O(N(p_1 + \cdots + p_n))$ дій при $N = p_1 p_2 \cdots p_n$, зокрема, при $N = 2^n$ знадобиться $O(N \log(N))$ дій.

Після того як ми розрахуємо спектрограму, можна визначити основну частоту, знайшовши індекс квадрата максимальної величини. Покращений алгоритм знаходить кілька таких місць, інтервалів частот-кандидатів, з величиною квадрата у верхній частині максимальних значень і далі аналізує їх для перевірки основних частот-кандидатів, використовуючи дані сигналу.

Коли на музичному інструменті грають ноту, звукові хвилі генеруються струнами, повітрям або динаміком – інструмент генерую музикальну ноту. Одним із показників музичної ноти є висота тону (основна частота). Зазвичай частоти музичного алфавіту поділяються на октави а далі на півтони. Октава

складається з 12 звуків, які називаються: C (простий), C#, D#, E, F, F#, G, G#, A, A# та B. Октава також має назви: велика, мала, однолінійна, дволінійна. лінійна і т.д. Стандартний тон" (однолінійний A або LA4) має основну частоту звукових хвиль 440 Гц. Частоти двох сусідніх нот відрізняються на 2 1/12, а частоти одних і тих самих нот у двох сусідніх октавах відрізняються на 2. Ноти та їх основні частот зображено в таблиці 2.1.

Таблиця 2.1 – Таблиця нот та їх основних частин.

Назва примітки	Традиційні назви октав (наукові), Гц			
	Відмінно (2)	Маленький (3)	Однорядний(4)	Дворядний(5)
C	65.40	130.81	261.62	523.25
C#	69.29	138.59	277.18	554.36
D	73.41	**146.83**	293.66	587.32
D#	77.78	155.56	311.12	622.25
E	**82.40**	164.81	**329.62**	659.25
F	87.30	174.61	349.22	698.45
F#	92.49	184.99	369.99	739.98
G	97.99	**195.99**	391.99	783.99
G#	103.82	207.65	415.30	830.60
A	**110.00**	220.00	440.00	880.00
A#	116.54	233.08	466.16	932.32
B	123.47	**246.94**	493.88	987.76

Типова (шестиструнна) гітара зазвичай відтворює тони від великої октави до дворядної октави. У таблиці жирним шрифтом виділено лади відкритих струн (E2, A2, D3, G3, B3 та E4).

Також для сприйняття повної картини настроювання гітар відзначмо наступні методи настройки інструмента.

Класичний метод настройки:

– Настройка першої струни проводитися за контрольним звуком – звуку камертона – або по звуку вже налаштованого музичного інструмента.

– По першій струні налаштовується друга струна, котра, будучи прижатою на п’ятому ладу, повинна звучати однаково з першою відкритою (неприжатою) струною.

– Третя струна, прижата на четвертому ладу, налаштовується по другій відкритій струні.

– Четверта струна, прижата на п’ятому ладу, налаштовується по третій відкритій струні.

– П’ята струна, прижата на п’ятому ладу, налаштовується по четвертій відкритій струні.

– Шоста струна, прижата на п’ятому ладу, налаштовується по п’ятій відкритій струні.

Камертон – інструмент для фіксації та відтворення еталонної висоти звуку, яка також має назву «камертон». В практиці використовуються дві настройки музичних інструментів, хористів та інші. Існують механічні, акустичні та електронні камертони. Камертон зображено на рисунку 2.3.

Рисунок 2.3 – Камертон.

Налаштування флажолетом:

Допомагає робити більш точну настройку, так як точність ладів не завжди достатньо.

– Налаштування першої струни проводитися за контрольним звуком – звуку камертона – або по звуку вже налаштованого музичного інструмента.

– Шоста струна налаштовується так, щоб її флажолет на п’ятому ладу звучав в унісон з першою струною.

– П’ята струна налаштовується так, щоб її флажолет на сьомому ладу звучав в унісон з першою струною, а далі трохи підтягується, щоб виникло биття частоти 0,372 Гц (одне биття за 2,7 секунди);

– Четверта струна налаштовується так, щоб її флажолет на сьомому ладу звучав в унісон з п’ятою струною на п’ятому ладу, а далі трохи підтягується, щоб виникло биття частоти 0,497 Гц (одне биття за 2,01 секунди);

– Третя струна налаштовується так, щоб її флажолет на сьомому ладу звучав в унісон з четвертою струною на п’ятому ладу, а далі трохи підтягується, щоб виникло биття частоти 0,664 Гц (одне биття за 1,51 секунди);

– Друга струна налаштовується так, щоб її флажолет на сьомому ладу звучав в унісон з першою струною на сьомому ладу, а далі трохи підтягується, щоб виникло биття частоти 1,12 Гц (одне биття за 0,9 секунди);

Якщо налаштовувати по камертону А(«ля») першої налаштовується п’ята струна (флажолет на п’ятому ладу в унісон с камертоном), в далі перша та шоста, потім четверта, третя та друга.

При гарному музичному слуху, накопив достатній опит, можливо обійтись і без флажолетів, ловивши биття обертонів в звучанні відкритих струн.

Так як натягування струн веде за собою деформацію корпусу гітари та розладу струн, рекомендується настроювати гітару в 2 – 3 ітерації, першу з яких можливо робити без темперації(без точного підрахунку частоти биття).

Для відображення повної роботи програмного продукту було створена UML діаграма. Ця діаграма є Універсальною мовою моделювання, тобто мова позначень або побудови діаграм, призначена для визначення, візуалізації і документування моделей зорієнтованих на об'єкти систем програмного забезпечення. Вона повністю показує роботу алгоритму та її функціональної частини. UML діаграма зображена на рисунку 2.4.

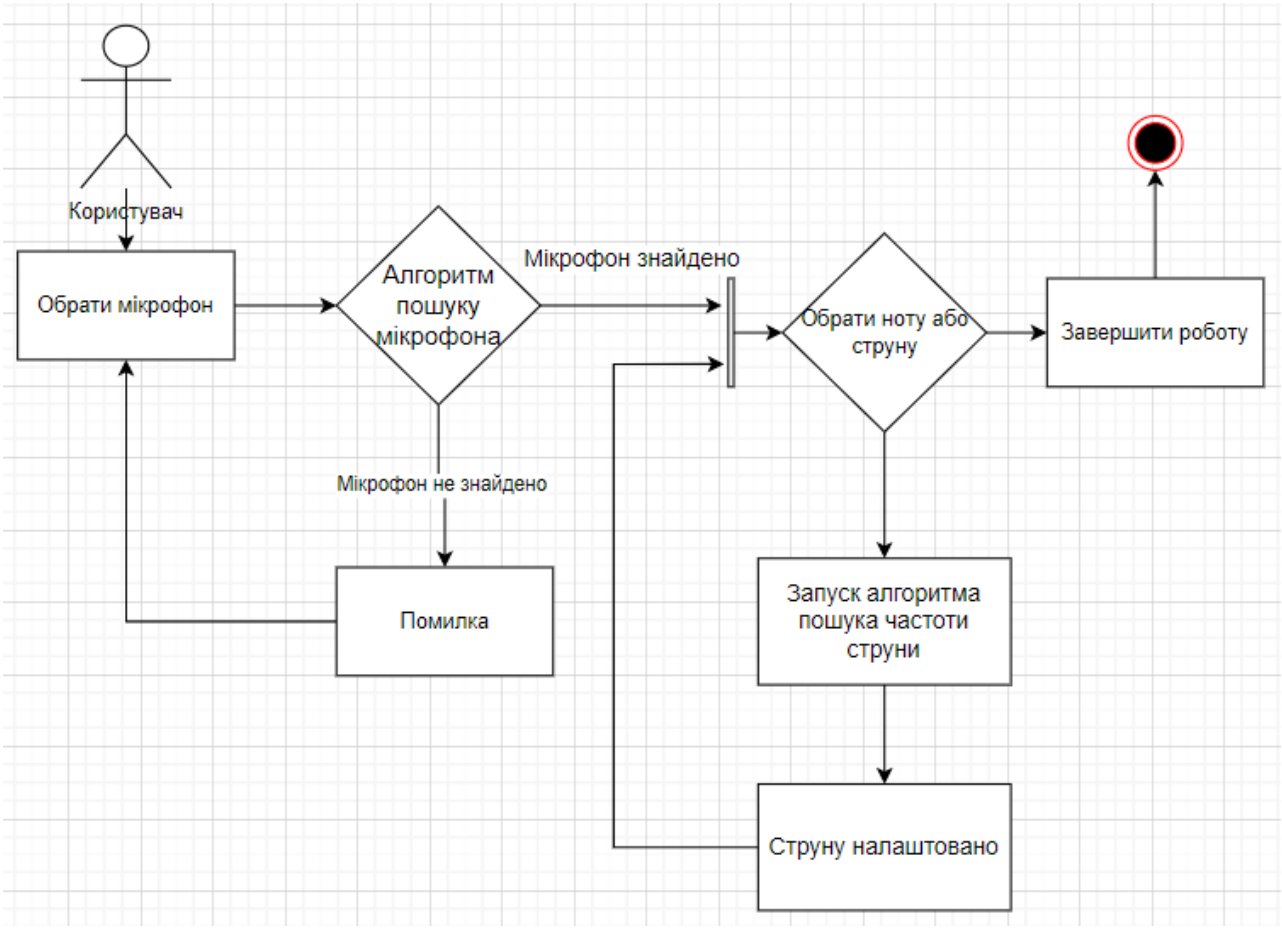


Рисунок 2.4 – UML діаграма тюнера.

Для розрахунку швидкого перетворення Фур'є використовувався алгоритм Кулі-Тьюкі. Це забезпечує хорошу продуктивність для виконання необхідного завдання. Для перевірки алгоритму програма аналізує приблизно 22 000 блоків зразків у реальному часі: аудіо записується з частотою 44 100 Гц та розміром вибірки 16 біт, а аналіз виконується двічі на секунду. Бібліотека аналізу звуку може бути представлена для виявлення тону, шуму фону, звуку або мови. Безперервна серія аудіоспектрограм також може відображатися у вигляді 2D (або 3D) зображення для візуального представлення. Тепер після розглянутого алгоритму Фур'є, аналізу її основних частин та сформованої UML діаграми, ми можем розробити діаграму алгоритму роботи програмної системи яка зображена на рисунку 2.5.

Рисунок 2.5 – Блок–сема алгоритму роботи програми

Словесний опис алгоритму:

1. Починається запуск та ініціалізація додатка;

2. Після запуска клієнт вибирає мікрофон для зчитування даних;

3. Після встановлення мікрофона запускаєм алгоритм програми через кнопку Start;

4. Зразу після цього починається зчитування аудіо даних;

5. Починається обробка даних які надходять з мікрофона, для визначення частоти звукової хвилі з найбільшою амплітудою(найгучніший звук) та зіставлення з найближчою нотою;

6. Відносно визначеному на попередньому кроку частоти, робиться висновок про те, яка саме струна звучала на даний момент.

7. На користувацькому інтерфейсі в реальному часі відображається інформація про поточну частоту та ймовірну струну;

8. Якщо частота звукової хвилі менше еталонної частоти передбачуваної струни, то відсилаємо команду стрілці інтерфейсу програми про рух вперед;

9. Якщо частота звукової хвилі більше еталонної частоти передбачуваної струни, то відсилаємо команду стрілці інтерфейсу програми про рух назад;

10. Якщо частота звукової хвилі лежить в діапазоні від еталонної частоти струни (f(s)) – допустиме відхилення частоти (fe) до f(s) + fe, то відправляємо сигнал стрілці інтерфейсу програми про наближення до центу шкали.

11. Відображаємо на користувацькому інтерфейсі про успішне налаштування струни та ноту.

12. Якщо користувач настроїв свій музичний пристрій, він натискає на кнопку Stop для завершення алгоритму.

13. Відправка сигналу на закінчення роботи алгоритму.

14. кінець роботи алгоритму.

2.4 Проектування інтерфейсу користувача

Важливим кроком у процесі проектування інтерфейсу користувача є аналіз дій користувача, які повинна забезпечити комп'ютерна програмна система. Не дізнавшись, що саме має робити система з погляду користувача, неможливо сформувати реалістичне уявлення про ефективний дизайн інтерфейсу.

Розробники інтерфейсів завжди повинні враховувати наступні особливості, такі як фізичні та розумові здібності людей, які працюватимуть із програмним забезпеченням. Люди протягом короткого часу можуть запам'ятати дуже обмежений обсяг інформації та припускаються помилок. Перерахуємо та опишемо основні принципи проектування інтерфейсу користувача:

– Врахування знань користувача – в інтерфейсі повинні використовуватися ті терміни та поняття, які зрозумілі користувачам. Іншими словами, інтерфейс повинен бути простим у використанні, щоб користувач міг звикнути до нього без особливих зусиль. Інтерфейс повинен використовувати зручні для користувача терміни, а об'єкти, керовані системою повинні бути безпосередньо пов'язані з робочим середовищем користувача;

– послідовність: той самий тип операції виконується однаковим чином;

– мінімальна кількість несподіванок – система має бути прогнозована;

– здатність до відновлення – інтерфейс повинен мати алгоритм або засоби відновлення даних після вчинення помилок;

– Підтримка користувачів – інструменти підтримки користувачів повинні бути інтегровані в інтерфейс програми. Вони повинні надавати різні рівні допомоги та базову інформацію на різних рівнях: від основ для новачків до повного опису можливостей програмної системи. Інтерфейс повинен надавати необхідну інформацію у разі помилок користувача та підтримувати контекстно-залежні засоби допомоги;

– підтвердження деструктивної дії – при виборі потенційно деструктивної дії користувач повинен ще раз підтвердити свій намір;

– можливість скасування – повернення системи у стан, у якому перебувала до застосування. Бажано підтримувати багаторівневе скасування, оскільки користувач не завжди зразу розуміє, що здійснив помилку;

– врахування різнорідності користувачів – інтерфейс має містити засоби зручної взаємодії з користувачами програмної системи, що мають різний рівень кваліфікації та різні можливості.

Розробник інтерфейсу користувача повинен вирішити два основні питання: як користувач буде вводити дані в систему і як ці дані будуть представлені користувачеві. Всі наразі існуючі сучасні інтерактивні системи, незалежно від їх призначення, використовують кольори. Правильне використання кольорів робить інтерфейс дуже зручнішим та привабливим для сприйняття. Якщо зробити навпаки то буде створено візуально непривабливий інтерфейс, що може провокувати помилки.

Користувачу який тільки завантажив додаток буває важко уявити, як він буде використовувати нове програмне забезпечення. Один зі способів подолання цієї проблеми полягає у використанні еволюційного методу розробки систем. Це означає, що користувачеві надають незавершену систему, яку потім змінюють і доповнюють доти, поки не буде з'ясовано й задоволено всі вимоги користувача. У якості альтернативи можна побудувати експериментальний прототип, який допоможе проаналізувати ці вимоги. І лише після цього створити систему.

Основний розділ системи – головний екран на якому зображена основна інформація. Загалом уся функціональна частина додатка представлена у вигляді одного вікна. Для оформлення було обрано три головних кольори: голубий, чорний та білий. Сам інтерфейс додатка повинен мати легкий для сприйняття та не перезавантажений деталями. Як фон було обрано ніжно голубий колір, а по середині екрана розміщено малюнок гітари та струни які потрібно настроювати. Під ним знаходяться шкала яка показує в яку сторону потрібно натягнути обрану струну. Зліва знаходяться дві кнопки які починають чи закінчують роботу тюнера та меню для вибору мікрофона, якщо ж натиснути старт та не вибрати мікрофон додаток видасть повідомлення про помилку, яке зображено на рисунку 2.6.

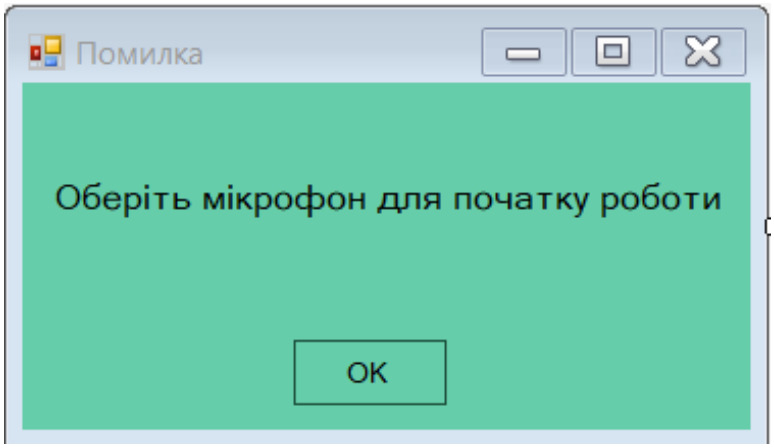


Рисунок 2.6 – Вікно для відображення помилки

Отже притримуючись всіх вище вказаних критеріїв та розподілення кольорів було розроблено прототип головної сторінки додатка (рисунок 2.7), на який буде виводитись весь функціонал додатка.

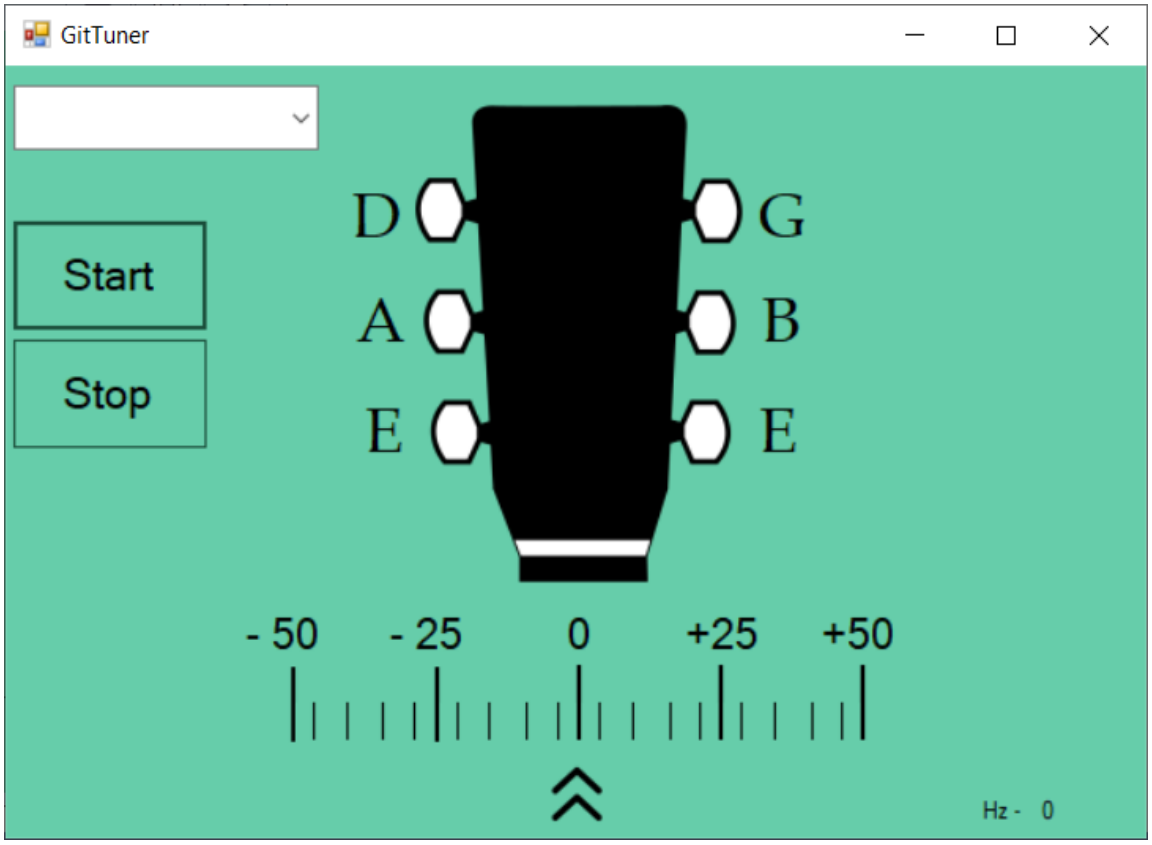


Рисунок 2.7 – Головне вікно програми

Результатом аналізу було визначено, що додаток буде розроблятися на MVVM архітектурі, оскільки це спростить реалізацію налаштувань та вкладок.

Також вибрана технологія реалізації додатка, що відповідає вимогам програмної системи, та доповнюють вибраний шаблон архітектурного проектування.

Отже в даному розділі було розглянуто архітектуру додатка сформований та показаний алгоритм дії програмної частини, розроблено зручний та сучасний інтерфейс який відповідає усім поточним тенденціям, а також обрано технологію та інструменти розробки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детальне проектування модулів

При розробці програмних модулів та проектування системи, будуть описані способи роботи цих модулів та способи взаємодії між собою. Буде розписана загальна структура, особливості архітектури та способи взаємодії між модулями програми. Також сформуємо діаграму класів. Розпочнемо з опису модулів та їх взаємодії між собою.

У попередньому розділі було спроектовано активності додатка. Так як кожна з них виконує оду функцію ми можемо наглядно сформулювати їх:

– Вибір мікрофона;

– обирання струни для налаштування;

– шкала для відображення точної настройки;

– кнопки для початку та завершення роботи;

В якості налаштування по замовчуванню був обраний класичний гітарний стрій. Також відомий як «іспанський» стрій або стрій E (мі). Найбільш популярний стрій, більшість табулатур та аплікатур дано саме для цього ладу. Класичний гітарний стрій зображено в таблиці 3.1.

Таблиця 3.1 – Класичний (“іспанський”) гітарний стрій

Струна	Нота	Частота(в герцах)
Перша	e^1 (мі першої октави)	329,63
Друга	b (сі малої октави)	246,94
Третя	g (соль малої октави)	196,00
Четверта	d (ре малої октави)	146,83
П’ята	A (ля великої октави)	110,00
Шоста	E (мі великої октави)	82,41

Розробимо діаграму класів, на якій показано клас GitTuner та структури даних які перебувають у відношенні асоціації (через свої поля), продемонстровано на рисунку 3.1.

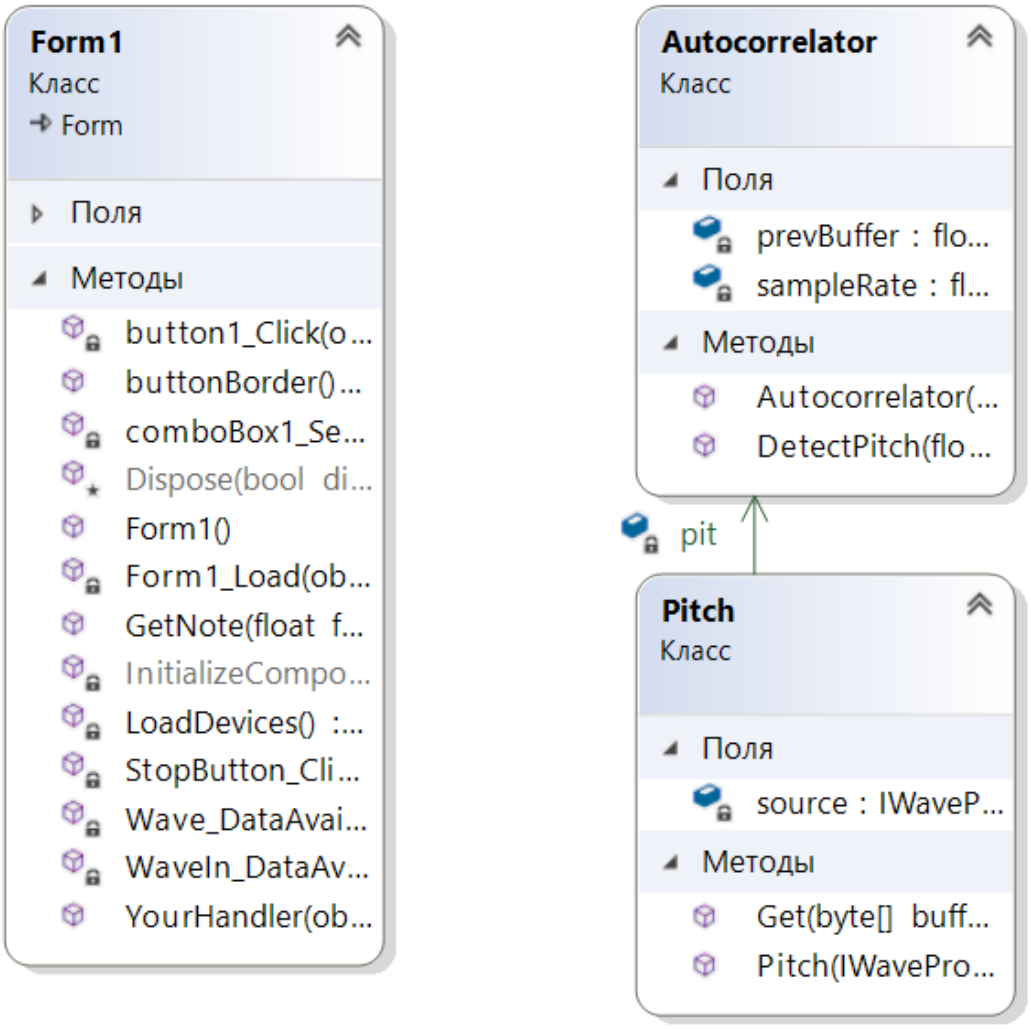


Рисунок 3.1 – Діаграма класів

3.2 Реалізація логіки додатка

Основною можливістю додатка є зчитування частоти та відображення положення цієї частоти, це і буде головною активністю конструктора форми. В коді головного класу в властивості кнопки button, яка починає роботу програми написана структура зчитування звуку. Також для цієї властивості визиваються класи які реалізують алгоритм Фур'є. Далі подано фрагмент реалізації:

```
     private void button1_Click(object sender, EventArgs e)
     {
         WaveInEvent wave = new WaveInEvent();
         wave.DeviceNumber = RecordingDeviceComboBox.SelectedIndex;
         wave.WaveFormat = new WaveFormat(44100, 1);
         wave.DataAvailable += WaveIn_DataAvailable;

         provider = new BufferedWaveProvider(wave.WaveFormat);

         // begin record
         wave.StartRecording();
         IWaveProvider stream = new Wave16ToFloatProvider(provider);
         Pitch pitch = new Pitch(stream);
         byte[] buffer = new byte[8192];

         int bytesRead;
         int n = 0;

         do
         {
          bytesRead = stream.Read(buffer, 0, buffer.Length);
          float freq = pitch.Get(buffer);
          if (freq != 0)
          {
           textBox1.Text += ("Freq: " + freq + " | Note: " +
GetNote(freq));
          }
         }
         while (n==1);

         wave.StopRecording();
         wave.Dispose();
     }
```

У вище вказаному методі код програми починається зі змінної WaveInEvent, вона визиває клас NAudio для подальшої роботи зі звуком, після чого з цього класа визивається метод DeviceNumber та присвоюється зміна яку ми вибрали у полі RecordingDeviceComboBox(поле для вибору мікрофона). Для того щоб була можливість вибирати мікрофон створено метод LoadDevices в якому через функції WaveIn.DeviceCount та метода GetCapabilities в поле RecordingDeviceComboBox виводиться список всіх доступних чи підключених мікрофонів до комп’ютера. Структура метода LoadDevices представлена нижче:

```
private void LoadDevices()
        {
            for (int devicesID = 0; devicesID < WaveIn.DeviceCount;
devicesID++)
            {
                var devicesInfo = WaveIn.GetCapabilities(devicesID);
```

```
RecordingDeviceComboBox.Items.Add(devicesInfo.ProductName);
            }
        }
```

Після чого встановлюється формат частоти та методу wave.DataAvailable присвоюється метод WaveIn_DataAvailable, який повинен у свою чергу відслідковувати чи буфер для запису не є пустим. Далі створюється так званий буфер потоку provider для запису вхідного аудіо. Так як ми вибрали мікрофон, визначили буфер для запису можемо починати запис вхідного звуку через метод wave.StartRecording(). В подальшому для налагодженої роботи системи яка буде працювати з аудіо яке надходить до звукової карти, створюються певні виключення. Весь звук який надходить з зовні повинен пройти багато перевірок, тому спочатку ми створимо зміну stream, в яку можемо передати аудіо яке надходить з звукової карти, та відправити його в окремо створений клас Pitch для цього, даний клас і буде містити ці виключення. Для наглядного прикладу продемонструємо його:

```
public class Pitch
    {
        IWaveProvider source;

        WaveBuffer waveBuffer;

        Autocorrelator pitchDetector;

        public Pitch(IWaveProvider source)
        {
            if (source.WaveFormat.SampleRate != 44100)
            {
                throw new ArgumentException("Джерело має знаходитись в
44.1kHz");
            }

            if (source.WaveFormat.Encoding !=
WaveFormatEncoding.IeeeFloat)
            {
                throw new ArgumentException("Джерелом мають бути
аудіодані IEEE з плаваючою комою ");
            }

            if (source.WaveFormat.Channels != 1)
            {
                throw new ArgumentException("Джерело звуку має бути моно
джерелом входу");
```

```
            }

            this.source = source;
            this.waveBuffer = new WaveBuffer(8192);
            this.pitchDetector = new
Autocorrelator(source.WaveFormat.SampleRate);
        }

        public float Get(byte[] buffer)
        {
            if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
            {
                waveBuffer = new WaveBuffer(buffer.Length);
            }

            int bytesRead = source.Read(waveBuffer, 0, buffer.Length);

            if (bytesRead > 0)
            {
                bytesRead = buffer.Length;
            }

            int frames = bytesRead / sizeof(float);

            return pitchDetector.DetectPitch(waveBuffer.FloatBuffer,
frames);
        }
    }
```

З приведеного коду бачимо наступні виключення які допомагають відсортувати вхідний потоку аудіо звуку, перша умова:

```
if (source.WaveFormat.SampleRate != 44100)
            {
                throw new ArgumentException("Джерело має знаходитись в
44.1kHz");
            }
```

Ця умова виводить на екран помилку, якщо джерело звуку вище за 44,1 кГц. Ця умова створена тому що існує частота дискретизації звука. Частота дискретизації це – частота запису або частота вибірки визначає кількості вибірок даних в одиницю часу, взятих з безперервного сигналу для подальшого оцифрування та зберігання пам'яті. При визначенні специфікації IP-аудіо щеплення враховує, чи повинен внутрішній PLL підтримувати дві частотні області, 16кГц-48кГц і 22,05кГц-44,1кГц. Підтримка цих двох частотних доменів збільшить площу PLL. Два типи перемикання частоти також повинні чекати на

блок PLL, і проміжна затримка становить близько 0,5 мс. Програмний пакет API також повинен підтримувати ці два типи частот одночасно, я не знаю, чому з'являється частота 22.05 44.1 кГц. Фактично 44,1кГц в основному використовується в mp3. Наступною умовою є:

```
if (source.WaveFormat.Encoding != WaveFormatEncoding.IeeeFloat)
            {
                throw new ArgumentException("Джерелом мають бути
аудіодані IEEE з плаваючою комою ");
            }
```

Тут зображена умова, яка допомагає перевірити що аудіодані IEEE з плаваючою комою. Число з плаваючою комою – це є форма представлення дійсних чисел, в якій число зберігається у формі мантиси і показника ступеня. При цьому число з плаваючою має фіксовану відносну точність і мінливу абсолютну. Найбільш часто використовується уявлення затверджено в стандарті IEEE 754. Число з плаваючою комою складається з:

– Мантиси(виражає значення числа без урахування порядку);

– знака мантиси(вказує на негативні чи позитивні числа);

– порядку(виражає ступень підстави числа, на яке повинно множитись мантиса);

– знака порядку.

Наглядно дане представлення зображено на рисунку 3.2.

Рисунок 3.2 – Рисунок числа з плаваючою комою

Цей стандарт може також використовуватись у програмних реалізаціях для арифметичних дій, так і в багатьох апаратних реалізаціях. Велика кількість

компіляторів, які призначені для різних мов програмування використовує саме цей стандарт для зберігання чисел та виконання математичних операцій.

Його також може визначати:

– Формати десяткових та двійкових значень з рухомою комою, для наступних обчислень та обміну даних;

– Всі арифметичні дії, такі як множення, ділення, додавання, віднімання, знаходження квадратного кореня та багато інших операцій

– Перетворення між форматами з рухомою комою та цілими форматами.

– Перетворення між різними форматами з рухомою комою.

– Перетворення між форматами з рухомою комою і символьними послідовностями.

– Виняткові ситуації в операціях з рухомою комою та їх обробку.

Та остатньою третьою умовою яка йде по списку стане, обробка джерела звуку яке має бути моно джерелом:

```
if (source.WaveFormat.Channels != 1)
            {
                throw new ArgumentException("Джерело звуку має бути моно
джерелом входу");
            }
```

Сам моно звук – це звук, призначений для прослуховування, ніби він виходив із однієї позиції. Зазвичай достатньо одного динаміка. Коли необхідно відтворити звук з кількох динаміків або навушників, ідентичні сигнали подаються на кожен гучномовець, що призводить до сприйняття одноканального звукового зображення в одному звуковому просторі між динаміками.

Відразу після умови використовується функція source, вона запису в себе виключення яке відбулось походу роботи додатка. Далі запис в буфер WaveBuffer та виклик наступного класу Autocorrelator. В цей клас ми передаємо значення source з класу Pitch яке ми щойно розглядали. Структура класу Autocorrelator:

```
class Autocorrelator
    {
        float[] prevBuffer;
        int minOffset;
        int maxOffset;
        float sampleRate;

        public Autocorrelator(int sampleRate)
        {
            this.sampleRate = (float)sampleRate;
            int minFreq = 75;
            int maxFreq = 335;

            this.maxOffset = sampleRate / minFreq;
            this.minOffset = sampleRate / maxFreq;
        }

        public float DetectPitch(float[] buffer, int frames)
        {
            if (prevBuffer == null)
            {
                prevBuffer = new float[frames];
            }
            float secCor = 0;
            int secLag = 0;

            float maxCorr = 0;
            int maxLag = 0;

            for (int lag = maxOffset; lag >= minOffset; lag--)
            {
                float corr = 0;
                for (int i = 0; i < frames; i++)
                {
                    int oldIndex = i - lag;
                    float sample = ((oldIndex < 0) ? prevBuffer[frames +
oldIndex] : buffer[oldIndex]);
                    corr += (sample * buffer[i]);
                }
                if (corr > maxCorr)
                {
                    maxCorr = corr;
                    maxLag = lag;
                }
                if (corr >= 0.9 * maxCorr)
                {
                    secCor = corr;
                    secLag = lag;
                }
            }
            for (int n = 0; n < frames; n++)
            {
                prevBuffer[n] = buffer[n];
            }
            float noiseThreshold = frames / 1000f;
            if (maxCorr < noiseThreshold || maxLag == 0) return 0.0f;
            return this.sampleRate / maxLag;
        }
    }
```

На самому початку класу водяться три зміні minOffset, maxOffset, sampleRate типу int і float, та масив prevBuffer. Далі конструктор класу приймає значення, яке ми передавали з Pitch в зміну sampleRate. Перейдемо далі до методу Get, який знаходиться в класі Pitch.cs:

```
public float Get(byte[] buffer)
        {
            if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
            {
                waveBuffer = new WaveBuffer(buffer.Length);
            }

            int bytesRead = source.Read(waveBuffer, 0, buffer.Length);

            if (bytesRead > 0)
            {
                bytesRead = buffer.Length;
            }

            int frames = bytesRead / sizeof(float);

            return pitchDetector.DetectPitch(waveBuffer.FloatBuffer,
frames);
        }
```

В представленому методі вхідне аудіо передається в waveBuffer, зміна bytesRead переводить в аудіо в байти, провіряє чи число більше нуля. Далі в локальну зміну frames поміщаємо математичну операцію bytesRead / sizeof(float). Оператор sizeof використовується для отримання кількості байт, що займаються змінною даного типу. Аргументом оператора sizeof має бути або ім'я sizeof, або параметр типу, якого застосовується обмеження, що вказує на те, що це некерований тип. Тут як бачимо оператором типу є float. В самому кінці метод повертає pitchDetector.DetectPitch(waveBuffer.FloatBuffer, frames). Метод DetectPitch і визначає тон аудіо яке надходить із звукової карти. Структура метода DetectPitch:

```
public float DetectPitch(float[] buffer, int frames)
        {
            if (prevBuffer == null)
            {
                prevBuffer = new float[frames];
            }
            float secCor = 0;
```

```
            int secLag = 0;

            float maxCorr = 0;
            int maxLag = 0;

            for (int lag = maxOffset; lag >= minOffset; lag--)
            {
                float corr = 0;
                for (int i = 0; i < frames; i++)
                {
                    int oldIndex = i - lag;
                    float sample = ((oldIndex < 0) ? prevBuffer[frames +
oldIndex] : buffer[oldIndex]);
                    corr += (sample * buffer[i]);
                }
                if (corr > maxCorr)
                {
                    maxCorr = corr;
                    maxLag = lag;
                }
                if (corr >= 0.9 * maxCorr)
                {
                    secCor = corr;
                    secLag = lag;
                }
            }
            for (int n = 0; n < frames; n++)
            {
                prevBuffer[n] = buffer[n];
            }
            float noiseThreshold = frames / 1000f;
            if (maxCorr < noiseThreshold || maxLag == 0) return 0.0f;
            return this.sampleRate / maxLag;
        }
    }
```

Саме в методі DetectPitch і розписаний алгоритм Фур'є, для визначення частоти. Давайте розберемо точніше фрагмент коду. В першому циклі for ми починаємо обчислення з низьких частот та поступово переходимо до вищих. В змінній corr типу даних float, ми обчислюємо як суму квадратів. Наступним кроком порівнюємо чи corr більше за maxCorr, у випадку коли більше, maxCorr ми присвоюємо значення corr, та значенню типу int maxLag = lag. В кінці алгоритм повертає значення частоти

Після того як звук пройшов усі умови можемо повернутись назад до циклу класу GitTuner. Далі створюється змінна buffer для зчитування байтів, та ще ода зміна bytesRead яка буде використовуватись в циклі do wile. У циклі do через цю змінну зчитуємо з буфера байти. Далі змінна freq типу float присвоюємо метод Get, та всередину всовуємо буфер з аудіо потоку. Провіряємо чи freq не дорівнює

нулю та через функцію GetNote(freq) виводимо в textbox ноти. Структура метода GetNote зображена нижче:

```
public string GetNote(float freq)
        {
            float baseFreq;

            foreach (var note in noteBaseFreqs)
            {
                baseFreq = note.Value;

                for (int i = 0; i < 9; i++)
                {
                    if ((freq >= baseFreq - 0.5) && (freq < baseFreq +
0.485) || (freq == baseFreq))
                    {
                        return note.Key + i;
                    }

                    baseFreq *= 2;
                }
            }

            return null;
        }
```

В методі створюємо цикл foreach, в якому берез значення з бібліотеки Dictionary типу string, під назвою noteBaseFreqs, потім берем значення елемента з масиву в baseFreq, та створюємо ще один цикл for. Демонстрація метода noteBaseFreqs зображена нижче:

```
Dictionary<string, float> noteBaseFreqs = new Dictionary<string, float>()
            {
                { "C", 16.35f },
                { "C#", 17.32f },
                { "D", 18.35f },
                { "Eb", 19.45f },
                { "E", 20.60f },
                { "F", 21.83f },
                { "F#", 23.12f },
                { "G", 24.50f },
                { "G#", 25.96f },
                { "A", 27.50f },
                { "Bb", 29.14f },
                { "B", 30.87f },
            };
}
```

Отже, ми розібрали алгоритм зчитування частоти звуку, які є класи та

методи для цієї реалізації. Які алгоритми нам допомагають та класи, на яких побудовані ті чи інші методи. Давайте тепер перейдемо до реалізації головного вікна додатку.

Головне вікно, являє собою комплекс різних користувацький графічних інтерфейсів, і включає в себе велику кількість різних елементів взаємодії та відображення інформації. Перерахуємо основні елементи:

– Кнопки старту роботи алгоритму та завершення, вибору ноти або струни та вихід з допоміжного вікна про помилку.

– Combo Box: для вибору мікрофона;

– Текстові поля: поле для виводу даних про частоту;

– Графічні елементи, такі як Picture Box для зображення даних для налаштування, стрілки та зображень.

Взаємодіяти можна з кожним елементом яких ми перерахували. Для створення симпатичних кнопок використовуємо наступний код:

```
public void buttonBorder()
        {
            button1.FlatAppearance.BorderSize = 0;
            button1.FlatStyle = FlatStyle.Flat;

            button2.FlatAppearance.BorderSize = 0;
            button2.FlatStyle = FlatStyle.Flat;

            button3.FlatAppearance.BorderSize = 0;
            button3.FlatStyle = FlatStyle.Flat;

            button4.FlatAppearance.BorderSize = 0;
            button4.FlatStyle = FlatStyle.Flat;

            button5.FlatAppearance.BorderSize = 0;
            button5.FlatStyle = FlatStyle.Flat;

            button6.FlatAppearance.BorderSize = 0;
            button6.FlatStyle = FlatStyle.Flat;
        }
```

Даний код забезпечує стилізацію кнопок для вибору струни, а саме заберає лінію для кращого вигляду. Кінцевий результат конструктора форми зображено на рисунку 3.3.

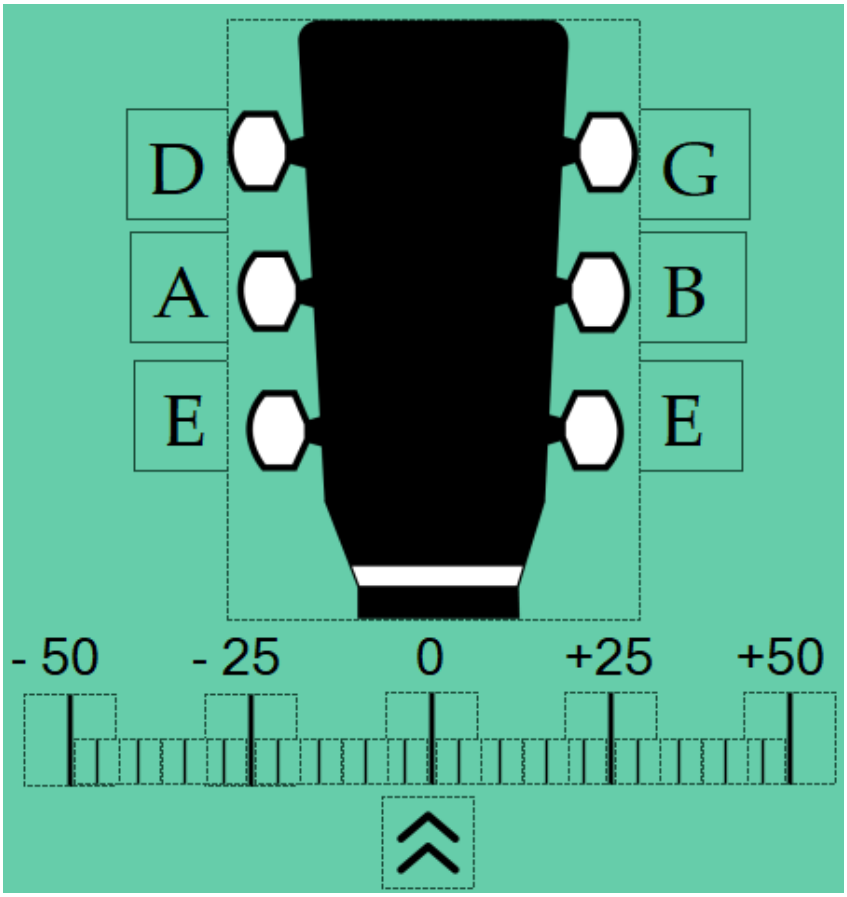


Рисунок 3.3 – Конструктор форми.

Повний код основних класів можна переглянути у додатку Б.

Отже, був розроблений настільний додаток, для налаштування гітар на платформі Windows. Була описана розробка програмного продукту, головного вікна та всіх його особливостей. Вигляд візуального інтерфейсу програми зображено в додатку И.

3.3 Керівництво користувача

Після того як ми розробили програмну систему, потрібно описати всі сценарії використання додатка, приклади користування додатком, опис роботи функціональної частини та перелічити умови при який додаток функціонуватиме правильно.

Ця програмна система використовує архітектуру MVVM і її основна функція це зчитування та обробка вхідного аудіо, тому для роботи додатка

непотрібно інтернет встановлення. Для функціонування всієї робочої частини достатньо тільки завантажити додаток. Ваш пристрій на якому встановлений додаток повинен лише відповідати мінімальним вимогам для роботи, щоб забезпечити комфортне користування. Опишемо основний сценарій роботи із додатком.

Отже, після завантаження заходимо в додаток, на головному екрані бачим три функціональні блоки з якими ми можемо контактувати. Перший це комбінований ящик, на якому висвітлюються всі доступні пристрої для зчитування аудіо, та можливість обрати відповідний пристрій. Другим елементом графічного інтерфейсу є кнопка Start, якщо мікрофон обрано вірно тоді починається робота додатку, тобто зчитування аудіо з обраного пристрою. Якщо ж відповідний пристрій не знайдено то вилазить вікно про помилку, у якому додаток нам говорить що не вдалося обрати відповідний мікрофон. Наступним елементом з яким ми можемо взаємодіяти це кнопка Exit, якщо ви налаштували свій музичний інструмент, то ця кнопка завершує роботу програмного продукту. Далі графічні елементи, та кнопки для обирання відповідної струни починають бути активними після успішного підключення мікрофона. Тобто обираємо до прикладу другу струну (B), та граємо на цій струні щоб побачити куди піде стрілка. Якщо стрілка пішла в плюс, це означає що струна перетягнута і потрібно її ослабити, щоб повзунок наблизився до нуля. Якщо ж навпаки після гри на струні повзунок пішов до мінуса, це означає що струна ослаблена і потрібно її підтягнути. Якщо струна знаходить свою частоту на екрані з’явіться назва цієї струни. Також додатково в правову нижньому куту виводиться частота з якою зараз б’ється струна. Результат на якому видно що струна налаштована правильно зображено на рисунку 3.4.

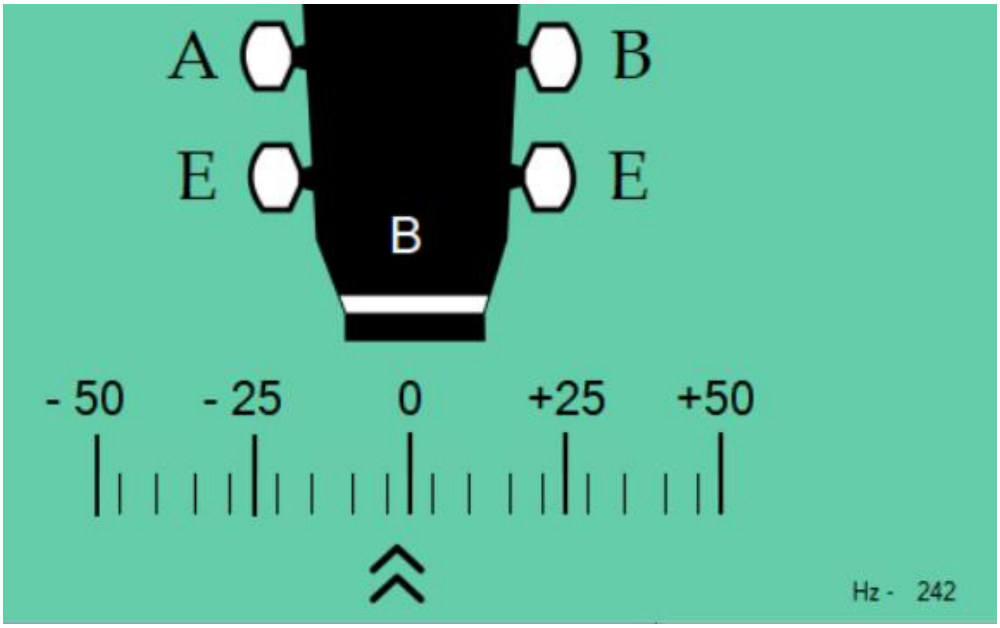


Рисунок 3.4 – Результат роботи.

3.4 Вимоги до технічних та програмних засобів

До вимог програмної системи зазначимо, що додаток буде працювати на пристроях з оперативною системою Windows. Тому, пристрої повинні відповідати мінімальному технічному забезпеченню, на яких дозволено використовувати ці операційні системи, після цього додаток буде працювати коректо.

Давайте опишемо мінімальні та рекомендовані системні вимоги для пристроїв які використовують оперативну систему Windows. Результат зображено в таблицях 3.2., 3.3.

Таблиця 3.2 – Мінімальні системні вимоги

Операційна система	Windows 7
Архітектура	32–bit
Тип системи	x32
Кількість оперативної пам’яті	2 ГБ
Вбудованої пам’яті	8 ГБ
Частота процесора	1.1 GHz

Таблиця 3.3 – Рекомендовані системні вимоги

Операційна система	Windows 8
Архітектура	64–bit
Тип системи	X64
Кількість оперативної пам’яті	4 ГБ
Вбудованої пам’яті	16 ГБ
Частота процесора	1.4 GHz

Отже, були описані всі технічні вимоги для додатка з оперативною системою Windows. Наглядно продемонстровано всі вимоги у таблицях наведених вище.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКА

4.1 Аналіз методів тестування додатка

Тестування програмного забезпечення – це метод перевірки відповідності фактичного програмного продукту вимогам та перевірки відсутності дефектів у програмному продукті. Вона включає в себе запуск компонентів програмного забезпечення/системи за допомогою ручних або автоматизованих інструментів для оцінки одного або декількох властивостей, що цікавлять. Метою тестування програмного забезпечення є виявлення помилок, прогалин або відсутніх вимог щодо фактичних вимог.

Проведення перевірки програми є важливою частиною розробки програмного забезпечення. Тестування повинно бути обов'язковою частина розробки системи, метою тестування продукту є:

– характеристики програмної системи, яка повинна показати відповідність між отриманою системою та яка заявлена;

– дослідити систему для визначення якості програмного продукту;

– спостерігання за додатком, перевірка вимог які заявлені, та перевірка реалізації заявлених функцій програмної системи.

– Також які є переваги тестування програмного забезпечення. Сформуємо наступні переваги використання додатків:

– Економічна ефективність: це є однією з найважливіших переваг тестування програм. Своєчасне тестування проекту може допомогти в заощадженні грошей в довгостроковій перспективі. Якщо виявити помилки на ранньому етапі їх виправлення може обійтись дешевше;

– Безпека: на етапі тестування ця стадія є найбільш вразлива і чутлива. Люди по своїй природі шукають надійні продукти, тому це допомагає усунути проблеми раніше;

– Якість продукту: ця вимога гарантує, що якісний продукт буде доставлений клієнтам;

– Задоволеність клієнта: найголовнішою метою програмного продукту – задоволення своїх клієнтів. Тому тестування забезпечує найкращий досвід користувача.

Тестування, може включати в себе аналіз та планування системи, розробку тестових сценаріїв роботи, отримання звіту роботи, проведення аналізу програми і виконання цих тестів. Ціль проведення складається в тому щоб підвищити якість програмної системи, виявити всі дефекти.

Давайте опишемо які наразі існують тестування додатка: тестування окремих компонентів програмної системи; тестування самих компонентів системи; тестування при отриманні. Види тестувань зображено на рисунку 4.1.

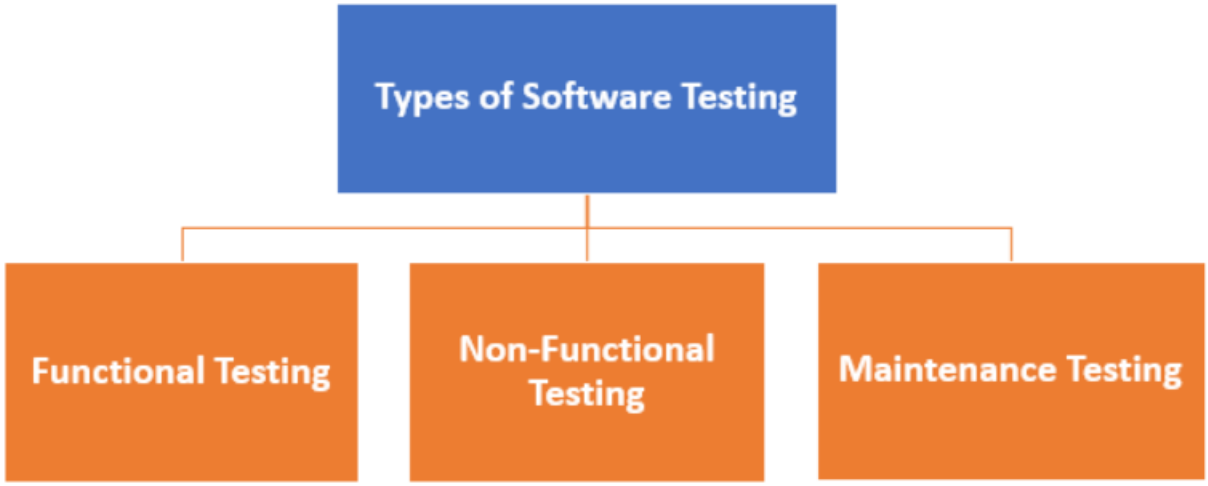


Рисунок 4.1 – Види тестувань

Після чого можемо починати більш детальний опис тестування, та розповісти про види тестувань які існують.

Модульне тестування – це тип тестування програмного забезпечення, при якому провіряються компоненти або окремі блоки додатка. Метою цього тесту полягає в тому, щоб підтвердити, що кожний елемент програмного коду працює як потрібно. Модульне тестування проводиться під час розробки (кодування) програми. Модульні тести виділяють частини коду та перевіряють його. Блоком може бути як метод, функція, модуль і так далі.

Наступним іде інтеграційне тестування, воно визначається як тип тестування, де програмні об’єкти тестуються як група. Однотипний проект може складатись з кількох програмних модулів, закодований різними програмістами. Метою є тестування та виявлення дефектів у взаємодії між модулями, коли вони є інтегрованими.

Системне тестування перевіряє код програмного продукту на помилки та несправності, може проводитись кооперування апаратної частини додатка.

Далі йде приймальне тестування, це є етап кінцевої перевірки програмної системи, який відповідає на запитання – чи відповідає система заявленим вимогам. Під час опису методів тестування ми вели мову про тести “Чорний ящик” – це тип тесту, який є принципом перевірки програмний продукт. Перевірка “Чорний ящик”, це є вид тестування який передбачає, те що ми не знаємо як буде працювати система.

Для додатка який ми розробили, та технологій розробки які ми обрали, найкраще підходять юніт – тести, ними можна протестувати окремі блоки коду програми, незалежно від інших модулів, та ці тести чудово вписуються в наш шаблон проектування MVVM. Для юніт – тестів характерними ознаками є:

– перевірка блоків коду програми, юніт – тести допомагають перевірити невелику частину коду програми, наприклад метод або класс. Проведення тестування над малими блоками дає змогу швидко написати багато тестів.

– проведення тестування окремих компонентів, моделей або представлень, дає можливість виявити помилку в різних компонентах системи.

– автоматичне тестування, тобто використання фреймворків для створення простих розвозів.

Так як ми обрали мову програмування c# і вибрали Visual Studio як середовище для розробки, доцільно використовувати наступні фреймворки:

– Ms Test: фреймворк який розробила компанія Microsof, для проведення юніт – тестів, яка підтримує . NET Core;

– NUnit: фреймворк який заснований на Junit, але портований на .NET;

Також хочу наголосити про нефункціональні тести, які можна провести для аналізу роботи системи: тестування для надійності додатка та тестування на сумісність із платформою. Тестування на надійність у розробленій системі є важливою частиною, оскільки робота покладається на захист програмної системи.

Отже, після опису мети повірки додатка, ми визначили ціль тестування та проаналізовані методи тестування програмної системи. Також були описані види тестування системи, вибір технології для проведення тестування і опис всіх доступних фреймворків. В результаті було обрано, що будуть проведені наступні методи тестування: модульне та інтеграційне.

4.2 Тестування додатку

Ми починаємо тестування компонентів програмного продукту з модульних тестів та перевірок: правильність роботи програми, правильність обробки вхідних та вихідних даних. Ми створимо таблиці із тестовими сценаріями, в яких будуть колонки: назва тестового сценарію; тестований модуль; метод модуля, що тестується; дані для входу до системи; очікуваний результат після завершення конкретного модуля. Потім ми проведемо функціональні тести системи, так само ми створимо таблиці функціональних вимог програми.

В таблиці 4.1 наглядно приведено основні тестові сценарії головної сторінки, це є методи та модулі, через які відбувається взаємодія

Таблиця 4.1 – Тестові сценарії головної сторінки настільного додатку

Назва сценарію	Модуль	Метод	Вхідні дані	Очікуваний результат
1	2	3	4	5
Отримання інформації про мікрофон для запису	GitTuner	LoadDevices	Пріоритет запиту	Назва пристрою
Вибір мікрофона	RecordButton_Load	DeviceNumber	Обраний мікрофон	Програма розпізнала мікрофон для запису

Кінець таблиці 4.3

Початок зчитування звуку	RecordButton_Load	StartRecording	Відсутні	Назва пристрою
Перевірка вхідного аудіо	Pitch	Pitch	Вхідний буфер	Відсортований звук
Обирання струни для налаштування	RecordButton_Load	StartButton	Відсутні	Обирання діапазону частоти для налаштування
Перевірка чи буфер не пустий	RecordButton_Load	WaveIn_DataAvailable	Вхідний буфер	Якщо буфер не пустий передача аудіо даних далі по потоку, якщо пустий
Переведення вхідного аудіо в масив байт	RecordButton_Load	buffer	Вхідний буфер	Перетворення масиву в масив байт для наступних операцій
Передача масива байт в метод Get для здійснення алгоритму Фур'є	Pitch	Get	Вхідний буфер	Перевірка масиву на довжину, та передача даних в наступний метод
Реалізація алгоритму Фур'є	Autocorrelator	DetectPitch	Вхідний буфер та масив байт	Успішна реалізація зчитування частоти з вхідного буфера
Визначення ноти	GitTuner	GetNote	Вхідна частота	Виведення ноти
Виведення на екран користувача результату	RecordButton_Load	freq	Вихідний буфер	Виведення резуфльтату частоти струн на графічний інтерфейс

Для функціонального тестування додатку використовувався ноутбук Lenovo ideapad 320 на операційній системі Windows 10. Для тестування деяких елементів графічного інтерфейсу програмної системи було автоматизовано шляхом використання фреймворку UITest.

Опис сценаріїв функціонального тестування додатка Cloudgram наведено у таблиці 4.2.

Таблиця 4.2 – Тестові сценарії основного функціоналу

Іденти-фікатор	Функціональна вимога	Вихідні дані	Очікуваний результат
1	2	3	4
CFT-01	Перегляд головної сторінки	Користувач відкриває головну сторінку додатка	Користувач бачить увесь графічний функціонал
CFT-02	Перегляд строки для вибору мікрофона	Користувач відкриває вікно в якому виводяться доступні пристрої для запису аудіо	Користувач бачить всі мікрофони
CFT-03	Натискання на кнопку Start	Запуска програмного інтерфейсу	Запуск інтерфейсу програми
CFT-04	Натискання на кнопку для налаштування ноти або струни	Користувач натискає на кнопку	Запуск алгоритму налаштування.
CFT-05	Ведення частоти струни на екран	Користувач бачить частоту струни	Виведення частоти
CFT-06	Натискання на кнопку Exit	Користувач натискає на кнопку	Завершення роботи програми

4.3 Аналіз результатів тестування

Результати модульного тестування програмного забезпечення наведено у таблиці 4.3.

Таблиця 4.3 – Результати модульного тестування

Назва сценарію	Метод	Вхідні дані	Отриманий результат	Результат
1	2	3	4	5
Отримання інформації про мікрофон для запису	GitTuner	LoadDevices	Пріоритет запиту	Правильно
Вибір мікрофона	RecordButton_Load	DeviceNumber	Обраний мікрофон	Правильно
Початок зчитування звуку	RecordButton_Load	StartRecording	Відсутні	Правильно
Перевірка вхідного аудіо	Pitch	Pitch	Вхідний буфер	Правильно
Обирання струни для налаштування	RecordButton_Load	StartButton	Відсутні	Правильно
Перевірка чи буфер не пустий	RecordButton_Load	WaveIn_DataAvailable	Вхідний буфер	Правильно

Кінець таблиці 4.3

1	2	3	4	5
Переведення вхідного аудіо в масив байт	RecordButton_Load	buffer	Вхідний буфер	Правильно
Передача масива байт в метод Get для здійснення алгоритму Фур'є	Pitch	Get	Вхідний буфер	Правильно
Реалізація алгоритму Фур'є	Autocorrelator	DetectPitch	Вхідний буфер та масив байт	Правильно
Визначення ноти	GitTuner	GetNote	Вхідна частота	Правильно
Виведення на екран користувача результату	RecordButton_Load	freq	Вихідний буфер	Правильно

Отже, в процесі написання четвертого розділу були визначені та описані основні методи проведення тестування додатків. Для написання модульних та інтеграційних тестів було обрано фреймворк за назвою NUnit,

Як видно після модульного тестування, дана програмна система працює саме так, як і передбачалося.

Результати функціональних тестів на реальних пристроях повністю задовольняють очікувані результати.

Виходячи із вказаного вище, можна зробити висновок що розроблений додаток працює саме так як і передбачалося та задовольняє вимоги, які були поставлені у технічному завданні.

ВИСНОВКИ

В процесі роботи та виконання дипломного проекту було проаналізовано предметну область, з'ясовані всі причини користуватись даним додатком, а також досліджені наявні програмні засоби для реалізації тюнера. В результаті було визначено переваги та недоліки, та було доведено потребу у розробці спеціалізованого програмного забезпечення, яке би призначатиметься саме для роботи з музичним інструментом. Також були сформовані вимоги які описані у технічному завданні та формувалися, опираючись на інші успішні додатки.

Далі ми провели аналіз переваг та недоліків найпопулярніших архітектур додатків, і встановлено, що для нашої розроблюваної системи найбільше підходить шаблон MVVM.

Окрім того, було проведено аналіз алгоритму налаштування гітари, що дало можливість визначити які саме дані та технології потрібно використовувати при розробці додатка, щоб досягнути цілей, поставлених у завданні до дипломного проектування.

У результаті детального аналізу було проведено наступний аналіз технологій і засобів, які будуть використовуватись в процесі розробки програмного забезпечення, визначено та описано основні модулі додатка і те, яким чином вони взаємодіють між собою. Також було прийнято рішення розробити певний функціонал для кращого і зручного налаштування музичного приладу.

На основі отриманих знань було реалізовано та здійснено опис компонентів бібліотеки NAudio та інших бібліотек. Розроблений в результаті програмний продукт було протестовано та доведено що він працює та задовольняє усім вимогам, які були поставлені до нього у технічному завданні.

В результаті виконання цього дипломного проекту було розроблено програмну систему для настроювання гітар для системи Windows, яка дозволяє її користувачам в автоматичному та ручному режимі налаштувати свій інструмент, а також допомогти початківцям тримати свою гітару завжди в налагодженому

стані.

Таким чином, завдання які були поставлені у процесі дипломного проектування були виконані, а мета роботи досягнута. Оскільки в процесі реалізації додатка використовувались новітні технології та засоби, отримані знання та вміння знадобляться у подальшій професійній діяльності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект : методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.
2. В. А. Шеховцов. Операційні системи. – К.: Видавнича група BHV, 2005. – 576с.
3. Carroll R. Bullet Journal [Електронний ресурс] / Ryder Carroll – Режим доступу до ресурсу: https://bulletjournal.com. (дата звернення – 10.03.2021).
4. Carroll, Ryder. The Bullet Journal method: track your past, order your present, plan your future. – New York: Portfolio/Penguin, 2018. – 310c.
5. My Diary [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=mydiary.journal.diary.diarywithlock.diaryjournal.secretdiary. (дата звернення – 17.03.2021).
6. Universum [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=ru.schustovd.diary. (дата звернення – 18.03.2021).
7. Pi Journal [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=diary.journal.app. (дата звернення – 18.03.2021).
8. SplenDO [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.splendapps.splendo. (дата звернення – 19.03.2021).
9. MyNotes [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=net.kreosoft.android.mynotes. (дата звернення – 19.03.2021).
10. Diary [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=journal.notebook.memoir.write.diary. (дата звернення – 20.03.2021).

11. Notes [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.gcteam.tonote. (дата звернення – 20.03.2021).

12. ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description

13. ISO/IEC 2382:2015, Information technology — Vocabulary — Part 1: Terms and definitions

14. C# documentation [Електронний ресурс] / MSDN. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/.

15. Windows Presentation Foundation documentation [Електронний ресурс] / MSDN. – Режим доступу до ресурсу: https://docs.microsoft.com/enus/dotnet/desktop/wpf/?view=netdesktop-5.0.

16. Прайс Марк Д. C# 7 та .NET Core. Крос-платформна розробка для професіоналів/Дж Прайс Марк. - Санкт-Петербург: Пітер, 2018. - 640 с.

17. Скит Д. C# для професіоналів: тонкощі програмування / Джон Скіт. - Москва: Вільямс, 2014. - 608 с.

18. Ріхтер Д. CLR via C#. Програмування на платформі Microsoft. Framework 4.0 мовою C# / Дж Ріхтер. - Санкт-Петербург: Пітер, 2012. - 928 с.

19. David B. M. TCP/IP Sockets in C#: Practical Guide for Programmers / B. M. David, J. D. Michael, L. C. Kenneth. – San Francisco: Elsevier, 2004. – 188 с.

20. Фейт С. Архітектура, протоколи, реалізація (включаючи IP версії 6 та IP Security) / Сідні Фейт. - Москва: Лорі, 2000. - 450 с.

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки програмної системи для налаштування гітар за допомогою засобів Windows Forms для ОС Windows. Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки додатка є «Завдання на дипломний проект», воно затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Програмна система для налаштування гітар з використанням засобів Windows Forms.

2.1 Функціональне призначення

Програма призначена для платформи Windows з метою надання зручного інструменту для налаштування гітар. Використовувати програмний продукт будуть користувачі комп’ютерів з операційною системою Windows.

2.2 Експлуатаційне призначення

Експлуатаційне призначення, програмна система може використовуватися на будь-якому пристрої з операційною системою Windows 8 та вище, попередньо встановивши цей додаток, додаткових налаштувань додаток не потребує, для початку роботи з ним.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Проводивши аналіз предметної області сформуємо перелік функціональних можливостей, які повинна надавати програма:

- Перегляд частоти звуку;
- можливість оприділяти ноти звука;
- можливість бачити правильність частоти струн.

Вимоги до інтерфейсу:

- Правильна робота візуальних частин;
- мінімальний і сучасний вигляд інтерфейсу;
- На головному екрані повинно бути візуальна частина яка показує правильність настроювання частоти струн;
- Спеціальне елемент для відображення частоти;

3.2 Вимоги до надійності

Програмна система «GitTuner» повинен виконувати наступні вимоги до надійності:

– Додаток має пройти юніт-тести, системні тести та інтеграційні тести

3.3 Умови експлуатації та вимоги до технічних засобів

Експлуатаційні умови повинні відповідати санітарним і технічним нормам експлуатації персонального комп’ютера, при температурі та відносній вологості навколишнього середовища, визначених для персональної обчислювальної техніки згідно з ГОСТ 15150-69. Програмне забезпечення буде використовуватись одним користувачем на своєму особистому пристрої.

Система повинна працювати на пристроях з версією Windows вище 7;

Мінімальна конфігурація системи:

– Windows 7;
– 2 гб ОП
– 300 Мб вільної пам'яті
– 32 – розрядна система;
– Конфігурація, що рекомендується:
– Windows 10;
– 8 гб ОП
– 320Мб вільної пам'яті
– 64 – розрядна система;

3.4 Вимоги до інформаційної та програмної сумісності

Програмний продукт повинен працювати під управлінням операційної системи Windows, яка повинна бути вище восьмої версії, а також схожих на неї системах від подібних виробників.

3.5 Спеціальні вимоги

Програма повинна мати зручний та приємний матеріальний дизайн інтерфейсу, зрозумілий для будь-якого користувача.

4 Вимоги до програмної документації

Програмна документація відповідно склад якої встановлено до ДСТУ 3008–95 та Єдиній системі програмної документації. Нижче наведено сам список програмних документів і їх зміст:

- структурна схема системи;
- текст самої програми – запис програмного продукту з необхідними поясненнями і коментарями;
- опис додатка – відомості про логічну і фізичну модель, відомості про функціонування додатка;
- програмна система і методика випробувань – вимоги, що підлягають відповідній перевірці при випробуванні додатка, також порядок і методи контролю;
- технічне завдання – цей документ;
- записка пояснення – схема самого алгоритму,
- загальний опис алгоритму або функціонування продукту програмної розробки, а також обґрунтування ухвалених технічних і техніко-економічних рішень;

5 Стадії та етапи розробки

Стадії та етапи розробки умовно позначеної системи проекту “ GitTuner ” подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадії розробки	Етапи робіт	Зміст робіт
Технічне завдання 02.01.22 – 31.01.22	Обґрунтування необхідності розробки програмного продукту	Коротка характеристика програмного продукту щодо організації самого тестування; основу та призначення розробки; вимоги до програмного комплексу та документація; стадії та етапи розробки додатка; порядок контролю та приймання
Ескізний проект 01.02.22 – 14.02.22	Розробка ескізного проекту	Розробка структури вхідних та вихідних даних; уточнення системи що буде розроблятися та середовища програмування; розробка і опис загальної алгоритмічної структури програмного забезпечення

Кінець таблиці А.1

Технічний проект 15.02.22 – 28.02.22.	Розробка технічного проекту додатка	Уточнення структури вхідних та вихідних даних, визначення форми їх подання; розробка детального алгоритму; розробка структури програмного забезпечення; остаточне визначення зміни технічних засобів; розробка заходів щодо впровадження програмного комплексу продукту
Робочий проект 01.03.22 – 10.04.22	Розробка програми	Реалізація програмного архітектури по тестуванню знань студентів; відладка; проведення розробки методики випробувань; проведення попередніх випробувань (тестування); коректування програмного забезпечення додатка; розробка документації
Розробка програмної документації 11.04.22 – 20.04.22	Розробка документації до програмного продукту	Розробка необхідної документації яка передбачена технічним завданням
Тестування системи 21.04.22 – 30.04.22	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення всіх основних тестів програми; коректування програмного забезпечення
Впровадження	Підготовка і передача програми.	Випуск додатку на платформі Microsoft Store; внесення коректувань в програмне забезпечення і документацію додатка

6 Порядок контролю та приймання

Здійснюватиметься контроль кінцевими користувачами системи, підключеними на етапі тестування програмної системи. Тільки як система розробки буде закінчена повинні бути проведене тестування на захист від некоректного введення.

ДОДАТОК Б

(Обов’язковий)

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

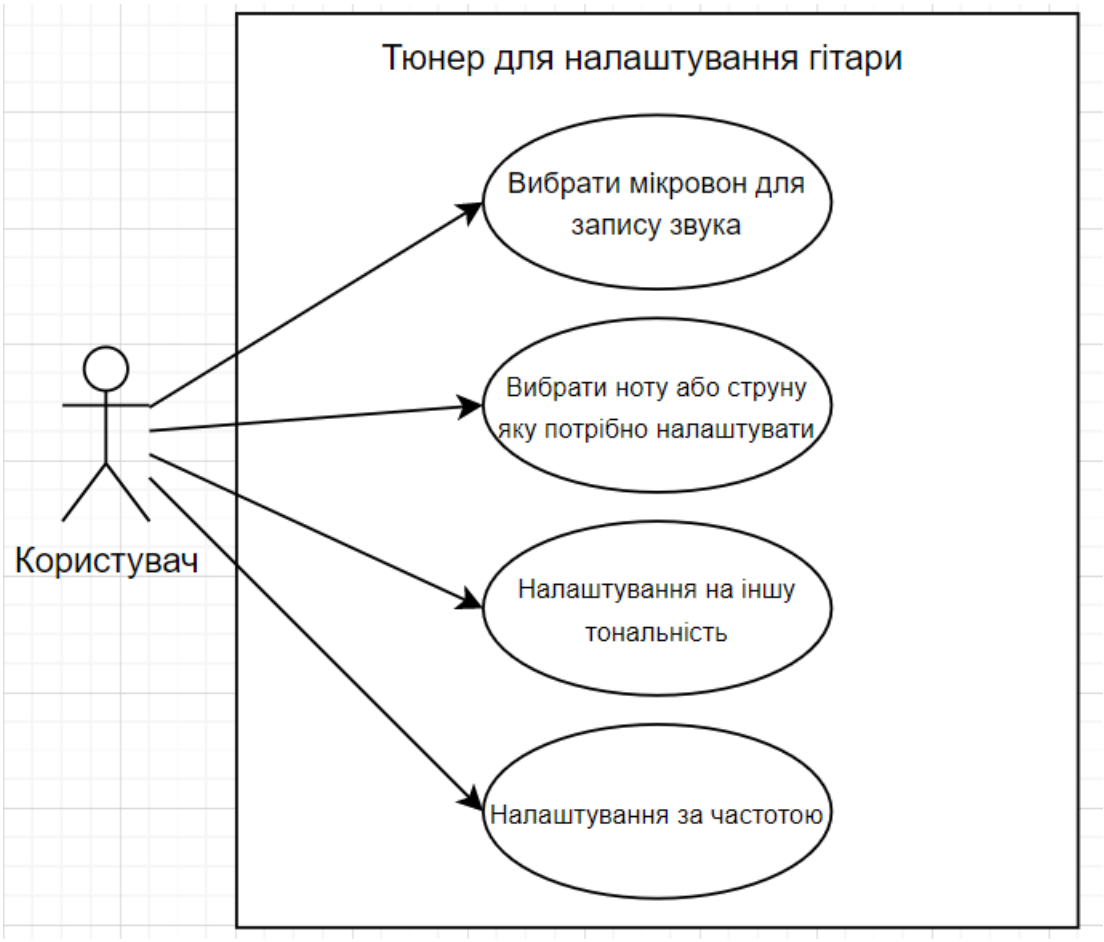


Рисунок Б.1 – Діаграма варіантів використання

ДОДАТОК В

(Обов’язковий)

UML ДІАГРАМА

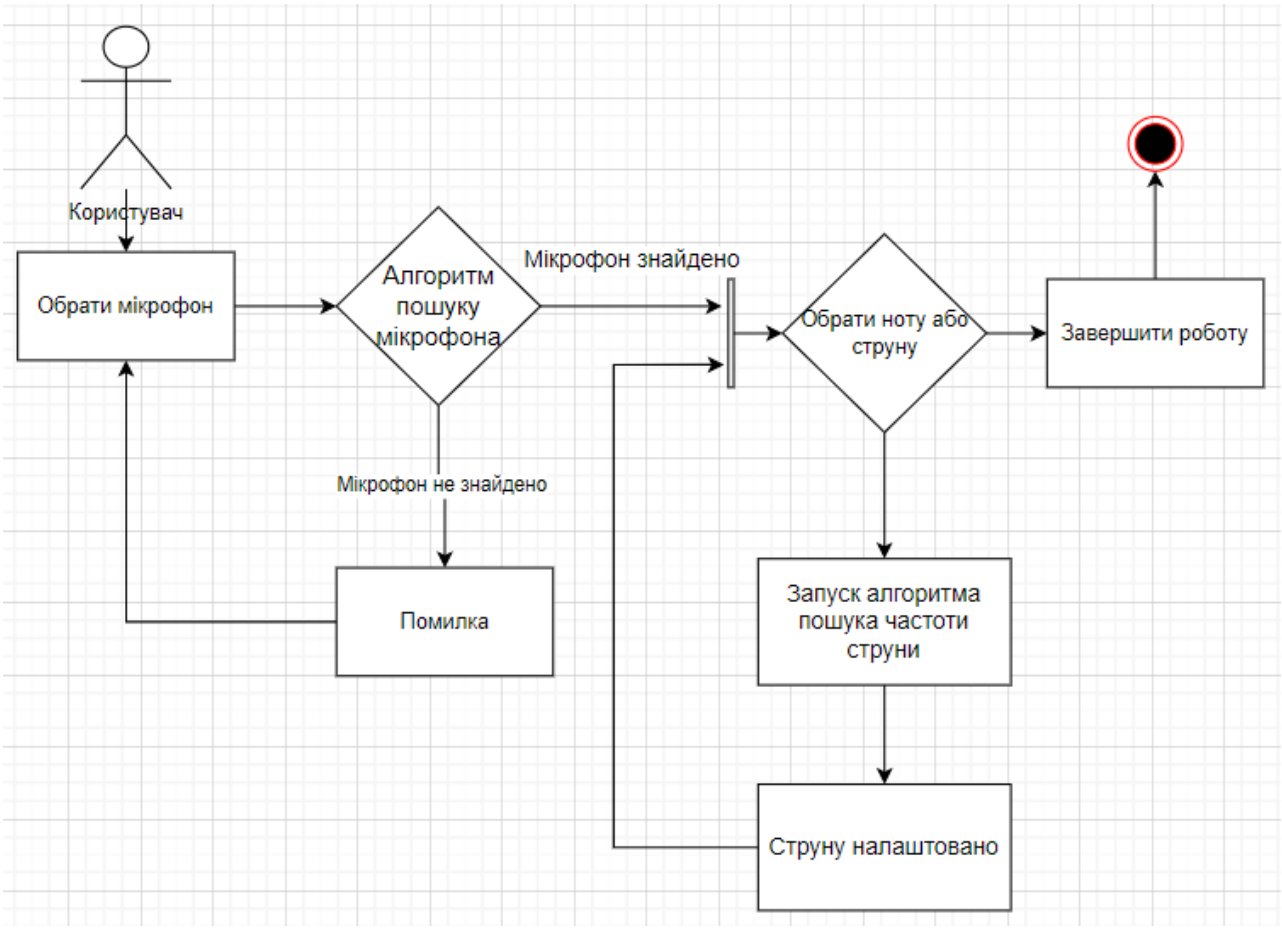


Рисунок В.1 – UML діаграма тюнера

ДОДАТОК Г

ДІАГРАМА АЛГОРИТМУ РОБОТИ

Рисунок Г.1 – Діаграма алгоритму роботи програми

ДОДАТОК Д

(Обов’язковий)

ДІАГРАМА КЛАСІВ

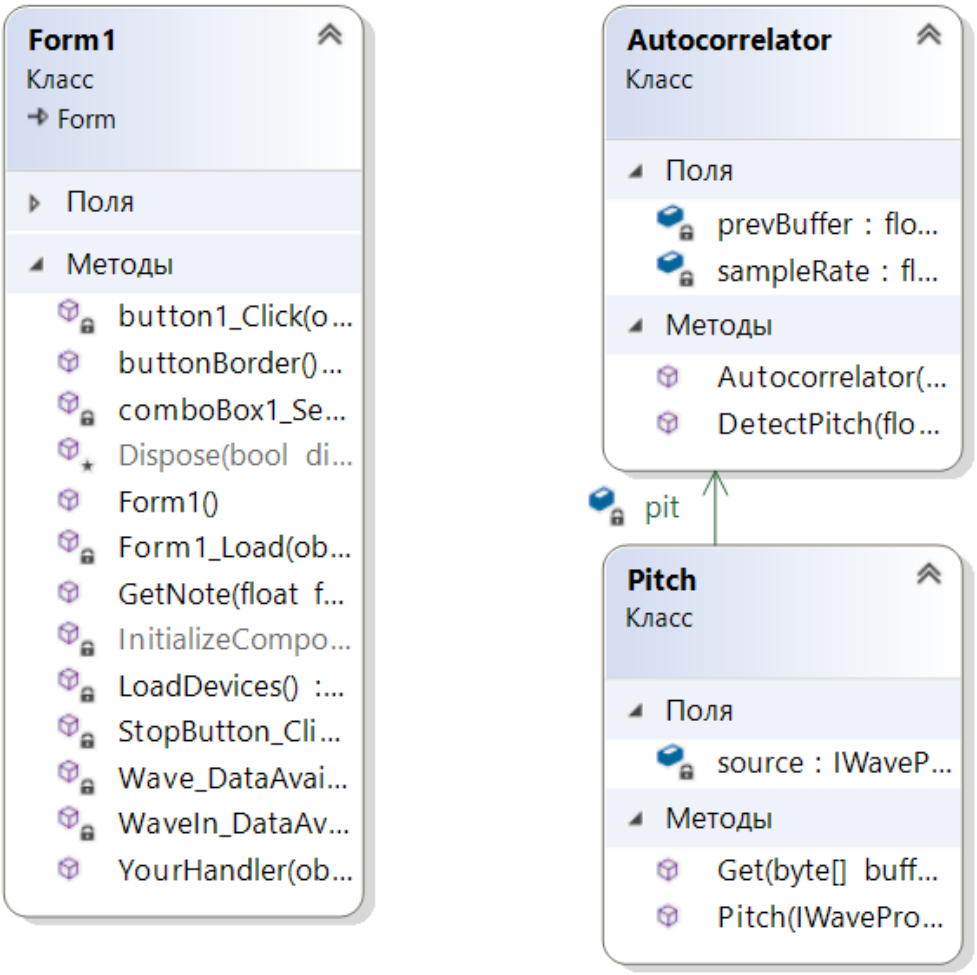


Рисунок Д.1 – Діаграма класів

ДОДАТОК И

(Обов’язковий)

ВИГЛЯД КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

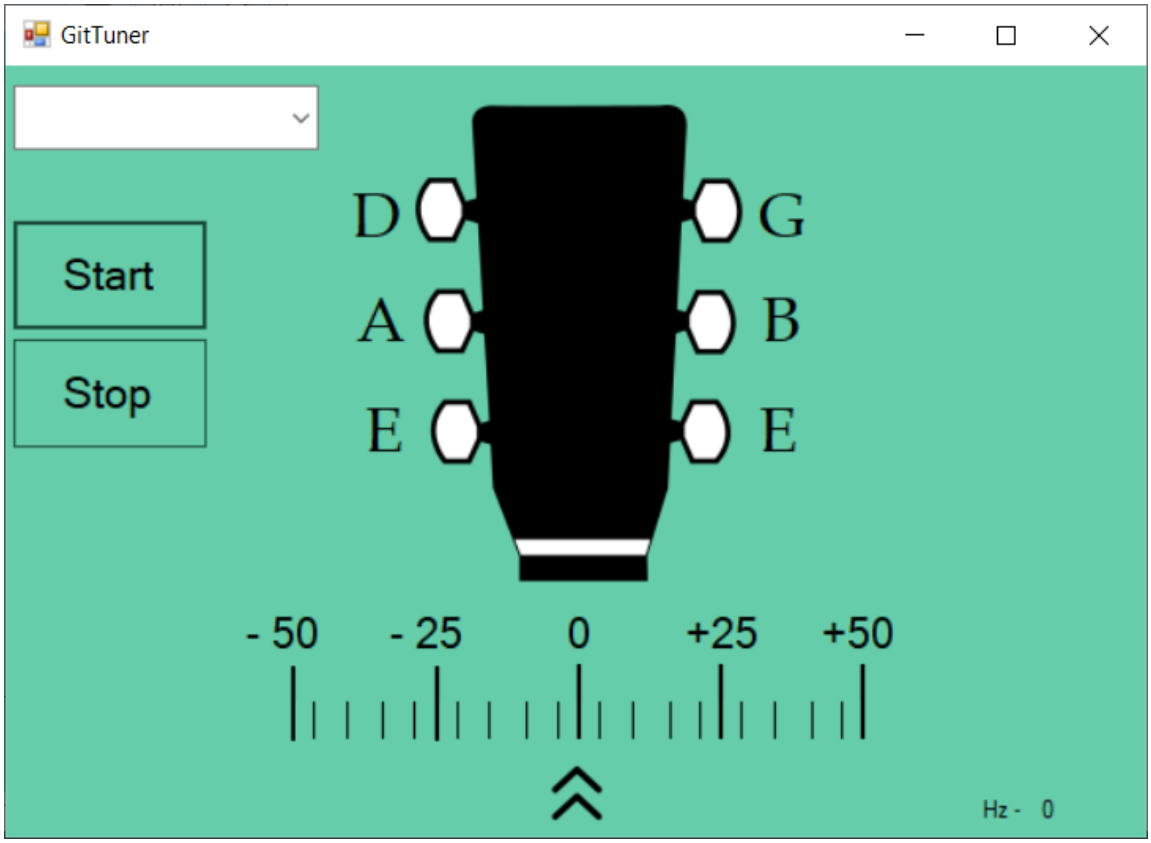


Рисунок И.1 – Головне вікно програми

ДОДАТОК Е
(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМНОЇ СИСТЕМИ «GitTuner»

Властивість кнопки RecordDutton:

```
private void button1_Click(object sender, EventArgs e)
        {
         WaveInEvent wave = new WaveInEvent();
         wave.DeviceNumber = RecordingDeviceComboBox.SelectedIndex;
         wave.WaveFormat = new WaveFormat(44100, 1);
         wave.DataAvailable += WaveIn_DataAvailable;
         provider = new BufferedWaveProvider(wave.WaveFormat);
         wave.StartRecording();
         IWaveProvider stream = new Wave16ToFloatProvider(provider);
         Pitch pitch = new Pitch(stream);
         byte[] buffer = new byte[8192];
         int bytesRead;
         int n = 0;
         do
         {
             bytesRead = stream.Read(buffer, 0, buffer.Length);
             float freq = pitch.Get(buffer);
             if (freq != 0)
             {
               textBox1.Text += ("Freq: " + freq + " | Note: " + GetNote(freq));
             }
         }
         while (n==1);
         wave.StopRecording();
         wave.Dispose();
}
```

Структура метода LoadDevices:

```
private void LoadDevices()
        {
            for (int devicesID = 0; devicesID < WaveIn.DeviceCount; devicesID++)
            {
                var devicesInfo = WaveIn.GetCapabilities(devicesID);
                RecordingDeviceComboBox.Items.Add(devicesInfo.ProductName);
            }
        }
```

Клас Pitch:

```
public class Pitch
    {
        IWaveProvider source;

        WaveBuffer waveBuffer;

        Autocorrelator pitchDetector;

        public Pitch(IWaveProvider source)
        {
            if (source.WaveFormat.SampleRate != 44100)
            {
```

```
                throw new ArgumentException("Джерело має знаходитись в
44.1kHz");
            }

            if (source.WaveFormat.Encoding != WaveFormatEncoding.IeeeFloat)
            {
                throw new ArgumentException("Джерелом мають бути аудіодані IEEE
з плаваючою комою ");
            }

            if (source.WaveFormat.Channels != 1)
            {
                throw new ArgumentException("Джерело звуку має бути моно
джерелом входу");
            }

            this.source = source;
            this.waveBuffer = new WaveBuffer(8192);
            this.pitchDetector = new
Autocorrelator(source.WaveFormat.SampleRate);
        }

        public float Get(byte[] buffer)
        {
            if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
            {
                waveBuffer = new WaveBuffer(buffer.Length);
            }

            int bytesRead = source.Read(waveBuffer, 0, buffer.Length);

            if (bytesRead > 0)
            {
                bytesRead = buffer.Length;
            }

            int frames = bytesRead / sizeof(float);

            return pitchDetector.DetectPitch(waveBuffer.FloatBuffer, frames);
        }
    }
```

Структура класа Autocorrelator:

```
class Autocorrelator
    {
        float[] prevBuffer;
        int minOffset;
        int maxOffset;
        float sampleRate;

        public Autocorrelator(int sampleRate)
        {
            this.sampleRate = (float)sampleRate;
            int minFreq = 75;
            int maxFreq = 335;

            this.maxOffset = sampleRate / minFreq;
            this.minOffset = sampleRate / maxFreq;
        }

        public float DetectPitch(float[] buffer, int frames)
```

```
        {
            if (prevBuffer == null)
            {
                prevBuffer = new float[frames];
            }
            float secCor = 0;
            int secLag = 0;

            float maxCorr = 0;
            int maxLag = 0;

            for (int lag = maxOffset; lag >= minOffset; lag--)
            {
                float corr = 0;
                for (int i = 0; i < frames; i++)
                {
                    int oldIndex = i - lag;
                    float sample = ((oldIndex < 0) ? prevBuffer[frames +
oldIndex] : buffer[oldIndex]);
                    corr += (sample * buffer[i]);
                }
                if (corr > maxCorr)
                {
                    maxCorr = corr;
                    maxLag = lag;
                }
                if (corr >= 0.9 * maxCorr)
                {
                    secCor = corr;
                    secLag = lag;
                }
            }
            for (int n = 0; n < frames; n++)
            {
                prevBuffer[n] = buffer[n];
            }
            float noiseThreshold = frames / 1000f;
            if (maxCorr < noiseThreshold || maxLag == 0) return 0.0f;
            return this.sampleRate / maxLag;
        }
    }
```

Метод Get:

```
public float Get(byte[] buffer)
        {
            if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
            {
                waveBuffer = new WaveBuffer(buffer.Length);
            }

            int bytesRead = source.Read(waveBuffer, 0, buffer.Length);

            if (bytesRead > 0)
            {
                bytesRead = buffer.Length;
            }

            int frames = bytesRead / sizeof(float);

            return pitchDetector.DetectPitch(waveBuffer.FloatBuffer, frames);
        }
```

Метода DetectPitch:

```
public float DetectPitch(float[] buffer, int frames)
        {
            if (prevBuffer == null)
            {
                prevBuffer = new float[frames];
            }
            float secCor = 0;
            int secLag = 0;

            float maxCorr = 0;
            int maxLag = 0;

            for (int lag = maxOffset; lag >= minOffset; lag--)
            {
                float corr = 0;
                for (int i = 0; i < frames; i++)
                {
                    int oldIndex = i - lag;
                    float sample = ((oldIndex < 0) ? prevBuffer[frames +
oldIndex] : buffer[oldIndex]);
                    corr += (sample * buffer[i]);
                }
                if (corr > maxCorr)
                {
                    maxCorr = corr;
                    maxLag = lag;
                }
                if (corr >= 0.9 * maxCorr)
                {
                    secCor = corr;
                    secLag = lag;
                }
            }
            for (int n = 0; n < frames; n++)
            {
                prevBuffer[n] = buffer[n];
            }
            float noiseThreshold = frames / 1000f;
            if (maxCorr < noiseThreshold || maxLag == 0) return 0.0f;
            return this.sampleRate / maxLag;
        }
    }
```

Структура метода GetNote:

```
public string GetNote(float freq)
        {
            float baseFreq;

            foreach (var note in noteBaseFreqs)
            {
                baseFreq = note.Value;

                for (int i = 0; i < 9; i++)
                {
                    if ((freq >= baseFreq - 0.5) && (freq < baseFreq + 0.485) ||
(freq == baseFreq))
                    {
```

```
                    return note.Key + i;
                }

                baseFreq *= 2;
            }
        }

        return null;
    }
```

метод noteBaseFreqs:

```
Dictionary<string, float> noteBaseFreqs = new Dictionary<string, float>()
            {
                { "C", 16.35f },
                { "C#", 17.32f },
                { "D", 18.35f },
                { "Eb", 19.45f },
                { "E", 20.60f },
                { "F", 21.83f },
                { "F#", 23.12f },
                { "G", 24.50f },
                { "G#", 25.96f },
                { "A", 27.50f },
                { "Bb", 29.14f },
                { "B", 30.87f },
            };
```

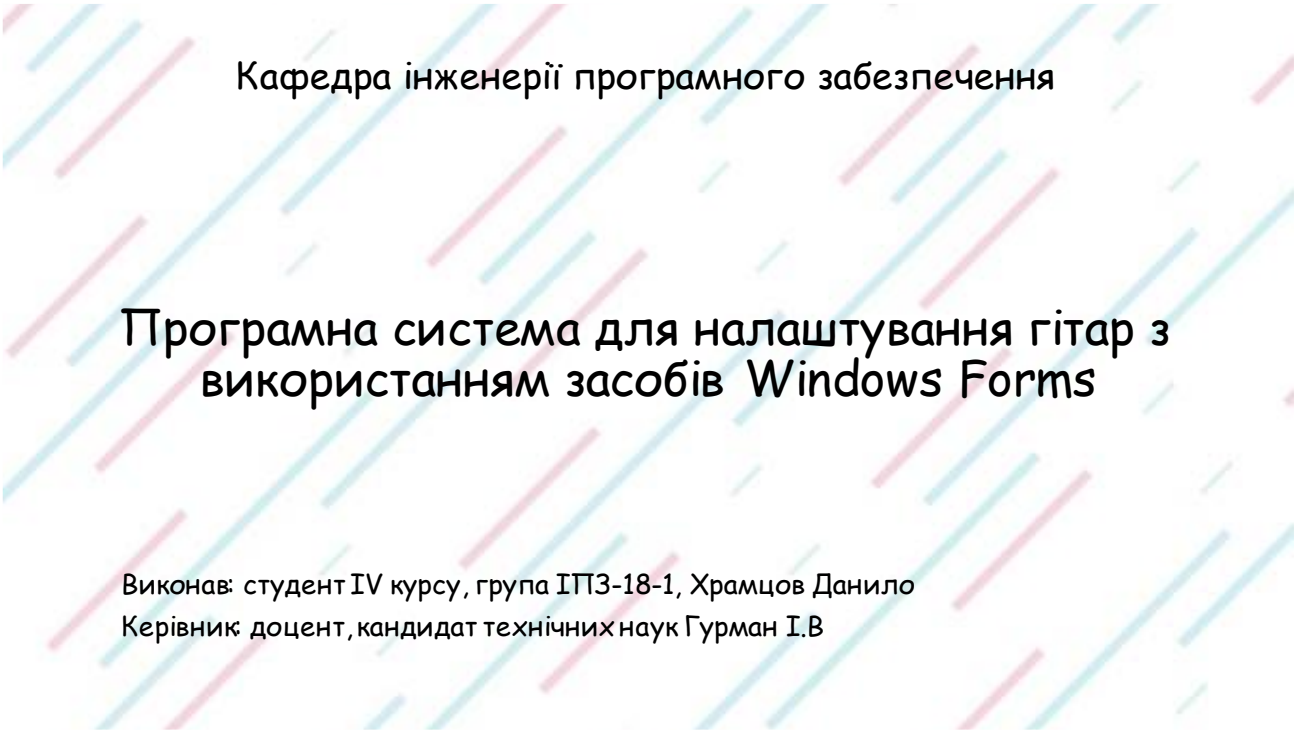
Метод ButtonBorder:

```
public void buttonBorder()
    {
        button1.FlatAppearance.BorderSize = 0;
        button1.FlatStyle = FlatStyle.Flat;

        button2.FlatAppearance.BorderSize = 0;
        button2.FlatStyle = FlatStyle.Flat;

        button3.FlatAppearance.BorderSize = 0;
        button3.FlatStyle = FlatStyle.Flat;

        button4.FlatAppearance.BorderSize = 0;
        button4.FlatStyle = FlatStyle.Flat;

        button5.FlatAppearance.BorderSize = 0;
        button5.FlatStyle = FlatStyle.Flat;

        button6.FlatAppearance.BorderSize = 0;
        button6.FlatStyle = FlatStyle.Flat;
    }
```

ДОДАТОК Ж

(Обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



Мета дослідження: розробка програмної системи для налаштування гітар з використання засобів Windows Forms, який би вирішував певні проблеми та допомагав у налаштуванні гітари.

Завдання дослідження:

- розробити максимально комфортний і зрозумілий інтерфейс для користувачів;
- переглянути наявні програмні продукти в цій галузі , та сформувати вимоги на розробку нового програмного забезпечення, яке буде відповідати потребам часу;
- розробити алгоритм зчитування частоти звуку(струн);
- виконати програмну реалізацію проекту;
- провести випробування та тестування програмного продукту;

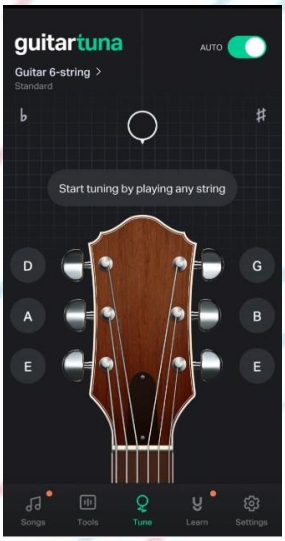
Актуальність теми

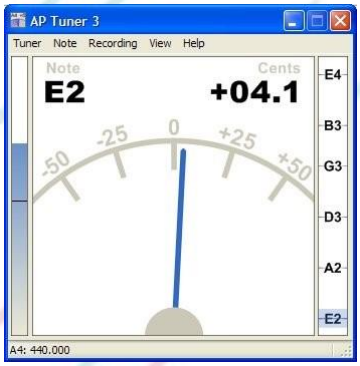
Актуальність теми дипломного проекту обумовлена необхідністю створення простого програмного продукту, у якому є повний функціонал без платних доповнень, та може повноцінно заміняти фізичний тюнер.

Також, даний програмний продукт повинен допомагати початківцям, у яких можуть виникнути серйозні перешкоди при налаштуванні гітари.

Наявні програмні продукти

інтерфейс додатку GuitarTuna





Інтерфейс додатку APGuitar tuner

Інтерфейс додатку PiJournal

Аналіз існуючих рішень

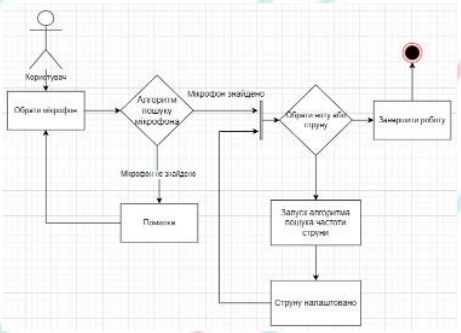
У більшості програм, які представлені на ринку - є два головних недоліка. Перший, та найголовніший недолік - це платний функціонал додатка, який дає змогу користуватись лише основними функціями, або знову ж присутня встроєна реклама, яка сильно дратує. Другим недоліком - є певна протилежність першому, це відсутність платного контенту та реклами, але знову малий функціонал, а іноді взагалі некоректна робота програми.

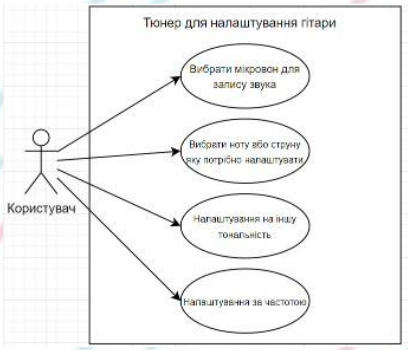
Проектування

Було виконано аналіз всіх існуючих шаблонів, та обрано найоптимальнішу архітектуру для додатка, а саме MVVM. Причиною вибору стала загальна продуктивність програм, що використовує цю модель.

Також для проектування було створено різні діаграми, для прикладу відзначимо UML діаграму та діаграму використання

UML діаграма





Діаграмавикористання

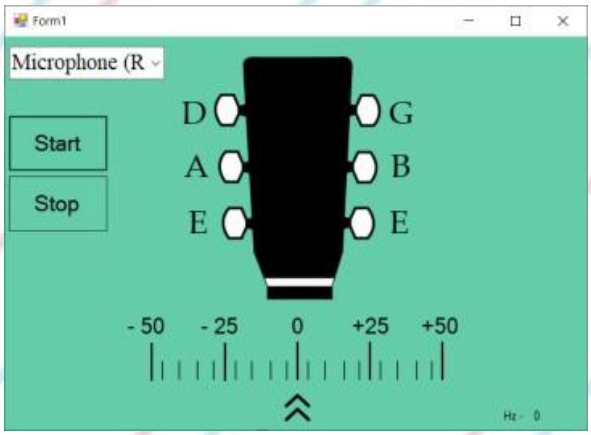
Інструменти та технології

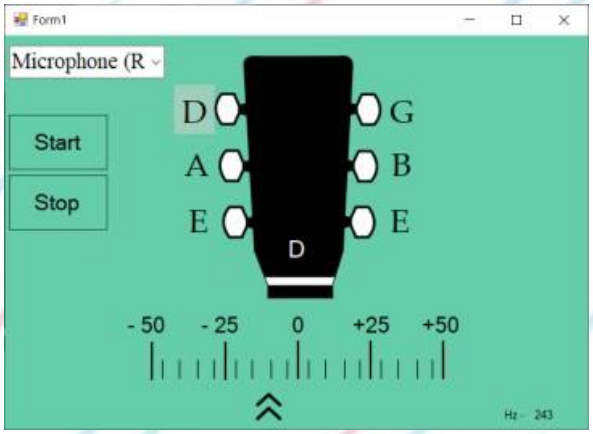
Для реалізації даного проекту було обрано мову програмування C#, на цій мові було розроблено весь функціонал.

Також, для створення функціональної частини, було використано аудіо бібліотеки, які працюють зі звуком, такі як:

- NAudio – це аудіо бібліотека .NET
- . Accord.NET Framework
- AForge.NET

Результат розробки





Отримані результати

Розроблений додаток дозволяє налаштовувати кожну із шести струн гітари, або зовсім перелаштувати на іншу тональність.

Також, додаток може оприділяти ноти, та виводити частоту струни.

Практична цінність отриманих результатів, полягає в успішній розробці програмної системи для налаштування гітар. Цю систему можуть використовувати початківці та професіонали для точного налаштування, без всякого платного контенту.

Завдяки даним особливостям, розроблена програмна система має високу конкурентну спроможність, в порівняні з існуючими додатками.

Висновки

В результаті виконання дипломного проекту, було проведено аналіз різних тюнерів для гітар, та їх особливостей, а також визначено недоліки існуючих рішень.

Розглянуто аналіз інструментів та технологій, для створення алгоритмів взаємодії зі звуком, і всіх функціональних частин. Опираючись на результати, розроблено архітектуру додатка, та спроектовано сучасний та функціональний інтерфейс.

Було створено саме програмне забезпечення, яке дозволить вирішити певні існуючи проблеми, та розроблено систему, яка відповідає визначеним функціональним вимогам.

Завідувачу кафедри інженерії програмного
забезпечення проф. Бедратюку Л. П.
здобувача вищої освіти Храмцова Д.І
Гурмана І.В.
факультет ІТ, 4 курс, група ІПЗ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.04.2022 р.
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 6.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

ID: 104376 Назва: Програмна система для налаштування гітар з використанням засобів Windows Forms Додано в БД: 2022-06-02 Автора: Д. І. Храмцов Керівники: І. В. Гурман Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	66017	613	8691 (13%)	95 (15%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1011429818

Дата перевірки:
02.06.2022 12:21:24 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2022 12:23:13 EEST

ID користувача:
100005589

Назва документа: **ДП_Храмцов.Д.І.(перевірка)docx**

Кількість сторінок: **65** Кількість слів: **10536** Кількість символів: **83251** Розмір файлу: **851.46 KB** ID файлу: **1011310159**

19.6%
Схожість

Найбільша схожість: **6.66% з джерелом з Бібліотеки** (ID файлу: **1008357188**)

10.4% Джерела з Інтернету 308 ..... Сторінка 67

12.2% Джерела з Бібліотеки 122 ..... Сторінка 69

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Храмцов Данило Ігорович

Тема Програмна система для налаштування гітар з використанням засобів Windows Forms

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 81

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було проаналізовано предметну область, а також визначено функційні вимоги до проектованого програмного забезпечення. Було здійснено аналіз відомого програмного забезпечення, наявного на ринку, розглянуто його переваги і недоліки та доведено актуальність розробки нового програмного забезпечення для налаштування гітар. Було проаналізовано технології для реалізації програмної системи та створено програмне забезпечення для налаштування гітар. Також було проведено тестування програмного забезпечення, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експлуатації.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто відомі рішення та визначені функційні вимоги до розроблюваного програмного забезпечення. У другому розділі представлено архітектуру та функціональну структуру додатку. Також здійснено вибір технологій для реалізації програмної системи, наведено алгоритм роботи проектованої системи та спроектовано інтерфейс користувача. У третьому розділі здійснено проектування модулів, реалізовано логіку додатків і наведено керівництво користувача та вимоги до проектованого засобу. В четвертому розділі було здійснено тестування програмної системи та проведено аналіз результатів тестування, в результаті чого було підтверджено коректну роботу програми.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки існує необхідність створення програмного забезпечення, у якому наявний повний функціонал для налаштування гітар, що надає можливість замінити фізичний тюнер.

5. Негативні сторони проекту _____

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів. _____

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований та послідовний, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал надає можливість наочно побачити деталі проектування системи. _____

8. Інші зауваження _____

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре». _____

РЕЦЕНЗЕНТ Бобровнікова Кіра Юліївна, к.т.н., доц. кафедри комп'ютерної інженерії та інформаційних систем _____

"6" червня 2022 р. _____ (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Програмна система для налаштування гітар з використанням засобів Windows Forms»

Автор: Храмцов Данило Ігорович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: освітньо-професійна програма Інженерія програмного забезпечення

Науковий керівник: Гурман Іван Васильович, к.т.н, доцент.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	Відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Сумарні співпадіння документа складають 19.6%

Серед співпадінь з документами у глобальній мережі максимальне співпадіння з одним документом становить 2,02% та стосуються переліку джерел посилань.

Серед співпадінь з документами з бібліотеки максимальне співпадіння з одним документом становить 6,66% та стосуються стандартних сторінок поясновальних записок тобто – титульний аркуш, бланк завдання, тощо.

Керівник _____ І. В. Гурман

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк