Міністерство освіти і науки України

Національний університет

«Львівська політехніка»

Кафедра електронних обчислювальних машин

**КУРСОВИЙ ПРОЄКТ**

з дисципліни “Системне програмне забезпечення”

на тему: “Розробка програмного забезпечення для керування звуковими пристроями. Розробка програми, що дозволяє керувати параметрами звукових пристроїв та виконувати різні дії з їх управлінням”

Студента 3-го курсу групи КІ-306

123 «Комп’ютерна інженерія»

Приймак Ю. О.

Керівник

Олексів М. В.

Національна шкала: \_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_

Оцінка ECTS: \_\_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Львів 2024

**ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ**

"ClearSound" - нова програма для справжніх меломанів! Завдяки цьому програмному забезпеченню можна насолоджуватися ідеальною якістю звуку і повністю контролювати звукові пристрої вашого комп'ютера.

## Функції, які має реалізувати програма:

1. Обробник подїї зміни гучності
2. Оновлення контролю гучності
3. Обробник події зміни значення контролю гучності
4. Обробник події тік таймера

# АНОТАЦІЯ

У даному курсовому проекті представлено розробку програмного забезпечення для керування звуковими пристроями. Основною метою цієї програми є забезпечення користувача якісним звуком при різних показниках гучності.

У роботі досліджено реалізацію програми з керування звуком в системі.

Описано процес розробки програми, включно із вибором засобів для реалізації

системи, структуру програмного забезпечення та реалізацію його функціональних можливостей.

Розроблена програма має зручний і зрозумілий для користувача інтерфейс, що вказує на простоту та доступність її використання.

# ЗМІСТ

[Функції, які має реалізувати програма: 2](#_Toc169260345)

[АНОТАЦІЯ 3](#_Toc169260346)

[ЗМІСТ 4](#_Toc169260347)

[ВСТУП 6](#_Toc169260348)

[РОЗДІЛ 1. Аналітичний огляд 7](#_Toc169260349)

[1.1 Аналіз відомих систем 7](#_Toc169260350)

[1.2 Аналіз методів вирішення 7](#_Toc169260351)

[1.3 Реалізація в моїй системі 8](#_Toc169260352)

[Висновок до розділу 9](#_Toc169260353)

[РОЗДІЛ 2. Проектування програми 10](#_Toc169260354)

[2.1 Обґрунтування розробки 10](#_Toc169260355)

[2.2 Специфічні алгоритми 10](#_Toc169260356)

[2.3 Структурна схема та її опис 11](#_Toc169260357)

[2.5 Вибір засобів для розробки програми 13](#_Toc169260358)

[2.6 Вибір технології 14](#_Toc169260359)

[Висновок до розділу 14](#_Toc169260360)

[РОЗДІЛ 3. Реалізація 14](#_Toc169260361)

[3.1 Інтерфейс користувача (UI) 14](#_Toc169260362)

[3.2 Вузли зі структурної схеми 15](#_Toc169260363)

[3.2.1 Інтерфейс користувача (UI) 15](#_Toc169260364)

[3.2.2 Модуль керування звуком 15](#_Toc169260365)

[3.2.3 Модуль роботи з аудіо пристроями 15](#_Toc169260366)

[3.2.4 Модуль обробки сигналів 15](#_Toc169260367)

[3.3 Алгоритми 16](#_Toc169260368)

[3.3.1 Алгоритм регулювання гучності 16](#_Toc169260369)

[3.3.2 Алгоритм застосування ефектів 16](#_Toc169260370)

[3.4. Діаграма класів 17](#_Toc169260371)

[3.5 Опис діаграми класів 17](#_Toc169260372)

[3.6 Формат файлів 18](#_Toc169260373)

[3.7 Модулі 18](#_Toc169260374)

[3.7.1 Модуль керування звуком 18](#_Toc169260375)

[3.7.2 Модуль роботи з аудіо пристроями 18](#_Toc169260376)

[3.7.4 Взаємодія між модулями 19](#_Toc169260377)

[3.8 Користувацький досвід (UX) 19](#_Toc169260378)

[3.9 Тестування 19](#_Toc169260379)

[3.10 Документація 20](#_Toc169260380)

[Висновок до розділу 20](#_Toc169260381)

[РОЗДІЛ 4. Тестування 20](#_Toc169260382)

[Висновок до розділу 23](#_Toc169260383)

[РОЗДІЛ 5. ВИСНОВКИ 24](#_Toc169260384)

[5.1 Розроблений додаток: 24](#_Toc169260385)

[5.2 Основні функції: 24](#_Toc169260386)

[5.3 Виконання завдання: 24](#_Toc169260387)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 25](#_Toc169260388)

[ДОДАТОК А 26](#_Toc169260389)

# ВСТУП

Сучасні комп'ютерні системи використовуються для виконання різноманітних завдань, серед яких важливу роль відіграє відтворення та обробка звуку. Від якості звуку залежить задоволення користувачів під час перегляду відео, прослуховування музики, гри в комп'ютерні ігри та під час використання мультимедійних додатків. Забезпечення високої якості звуку та зручного керування аудіо параметрами є важливими аспектами для користувачів, системних адміністраторів та розробників програмного забезпечення.

Основна проблема, з якою стикаються користувачі сучасних комп'ютерних систем, полягає в обмежених можливостях стандартних інструментів для керування звуком. Часто вбудовані засоби керування звуком, що надаються операційними системами, мають обмежений функціонал та незручний інтерфейс. Це особливо актуально для користувачів, які потребують точного налаштування звуку для професійних цілей, таких як аудіо інженери, музиканти або розробники мультимедійних додатків.

Для вирішення цієї проблеми існує потреба у розробці програмного забезпечення, яке б надавало можливості керування звуком, зручний та інтуїтивно зрозумілий інтерфейс, а також інтеграцію з різними звуковими пристроями.

Метою даної курсової роботи є розробка програмного забезпечення для керування звуком на комп'ютерах з операційною системою Windows. Програма буде надавати користувачам можливість отримувати дані про стан аудіо пристроїв, налаштовувати рівень гучності. У процесі розробки програми будуть використані сучасні методи та технології системного програмування, що дозволить забезпечити високу продуктивність та надійність роботи програми.

# РОЗДІЛ 1. Аналітичний огляд

## 1.1 Аналіз відомих систем

1. **Windows Volume Mixer**
   * ***Опис:*** Вбудований інструмент операційної системи Windows для керування гучністю окремих додатків та системних звуків.
   * ***Переваги:*** Інтуїтивно зрозумілий інтерфейс, легкість доступу, вбудований в операційну систему.
   * ***Недоліки:*** Обмежені функції, неможливість налаштування звукових ефектів, відсутність підтримки роботи з кількома аудіо пристроями одночасно.
2. **EarTrumpet**
   * ***Опис:*** Сторонній додаток для Windows, що розширює функціональність стандартного мікшера гучності.
   * ***Переваги:*** Розширені можливості налаштування гучності для окремих додатків, зручний інтерфейс, легкість використання.
   * ***Недоліки:*** Обмежена підтримка звукових ефектів, відсутність інтеграції з професійними аудіо пристроями, не підтримує детальні налаштування звукових параметрів.
3. **Voicemeeter**
   * ***Опис:*** Професійний віртуальний аудіо мікшер для Windows, що дозволяє керувати кількома аудіо пристроями одночасно.
   * ***Переваги:*** Потужні можливості для налаштування звуку, підтримка багатоканального аудіо, застосування звукових ефектів.
   * ***Недоліки:*** Складний інтерфейс, висока крива навчання, вимагає значних системних ресурсів.

## **1.2 Аналіз методів вирішення**

1. **Пряме налаштування гучності**
   * ***Опис:*** Використання API операційної системи для налаштування рівня гучності аудіо пристроїв.
   * ***Переваги:*** Простота реалізації, висока продуктивність.
   * ***Недоліки:*** Обмежена функціональність, неможливість застосування звукових ефектів.
2. **Віртуальні аудіо драйвери**
   * ***Опис:*** Використання віртуальних драйверів для створення віртуальних аудіо пристроїв, через які проходить аудіо сигнал.
   * ***Переваги:*** Розширені можливості для налаштування звуку, підтримка звукових ефектів.
   * ***Недоліки:*** Складність реалізації, високі вимоги до системних ресурсів, можлива затримка звуку.
3. **Мережеве керування аудіо**
   * ***Опис:*** Використання мережевих протоколів для керування аудіо пристроями через мережу.
   * ***Переваги:*** Можливість дистанційного керування, інтеграція з іншими мережевими системами.
   * ***Недоліки:*** Складність налаштування, висока залежність від мережевої інфраструктури, можлива затримка звуку.

1.3 Реалізація в моїй системі

Для створення конкурентоспроможного продукту, моя система буде поєднувати переваги наведених вище підходів і систем. Вона включатиме:

* Інтуїтивно зрозумілий інтерфейс для легкого налаштування гучності та перемикання між аудіо пристроями.
* Підтримку детального налаштування звукових параметрів.
* Інтеграцію з API операційної системи для забезпечення високої продуктивності.
* Використання віртуальних аудіо драйверів для розширення функціональності.

## Висновок до розділу

Проаналізувавши існуючі системи та методи керування звуком, можна зробити висновок, що кожна з них має свої переваги та недоліки. Моя система буде включати в себе найкращі аспекти цих рішень, забезпечуючи користувачам зручний інтерфейс, можливості налаштування звуку та високу продуктивність. Це дозволить створити конкурентоспроможний продукт, який задовольнить потреби як звичайних користувачів, так і професіоналів у сфері аудіо.

# РОЗДІЛ 2. Проектування програми

## 2.1 Обґрунтування розробки

Розробка програмного забезпечення для керування звуком на комп'ютерах з операційною системою Windows має на меті створення інструменту, який буде зручним та ефективним для користувачів. Основні вимоги до системи включають:

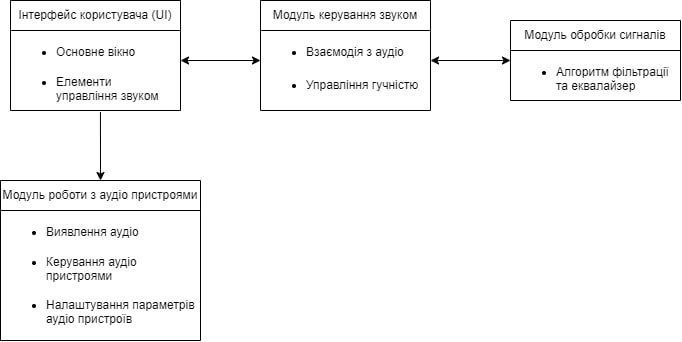
1. **Інтуїтивно зрозумілий інтерфейс:**
   * Зручність використання навіть для недосвідчених користувачів.
   * Легкий доступ до основних функцій налаштування звуку.
2. **Можливість детального налаштування гучності:**
   * Регулювання гучності як для системи в цілому, так і для окремих додатків.
3. **Висока продуктивність та надійність:**
   * Мінімальне використання системних ресурсів.
   * Стабільна робота без збоїв та зависань.

## 2.2 Специфічні алгоритми

Для забезпечення ефективної роботи програмного забезпечення будуть використані наступні алгоритми:

1. **Алгоритм регулювання гучності:**
   * Використання API операційної системи Windows для налаштування рівня гучності аудіо пристроїв.
   * Взаємодія з аудіо драйверами для забезпечення точного контролю гучності.

2.3 Структурна схема та її опис



*Рис.2.1. Структурна схема програми*

Структурна схема програмного забезпечення

Система складається з чотирьох основних модулів, які взаємодіють між собою для забезпечення повного контролю та обробки звуку на комп'ютерах з операційною системою Windows. Розглянемо роботу кожного модуля окремо:

1. Інтерфейс користувача (UI)

Основні функції:

* Основне вікно програми: забезпечує користувача доступом до всіх функцій програми.
* Елементи управління звуком: дозволяють користувачу регулювати г учність та налаштовувати інші параметри звуку.

Взаємодія:

* Передає команди до модуля керування звуком.
* Отримує інформацію від модуля керування звуком для оновлення інтерфейсу відповідно до поточного стану звуку.

2. Модуль керування звуком

Основні функції:

* Взаємодія з аудіо: обробляє команди від інтерфейсу користувача і передає їх до відповідних модулів.
* Управління гучністю: регулює рівень гучності відповідно до команд від інтерфейсу користувача.

Взаємодія:

* Отримує команди від інтерфейсу користувача.
* Передає команди до модуля обробки сигналів для фільтрації та еквалізації.
* Передає команди до модуля роботи з аудіо пристроями для керування підключеними пристроями.

3. Модуль обробки сигналів

Основні функції:

* Алгоритми фільтрації та еквалайзера: застосовує фільтри та еквалайзер до аудіо сигналів для покращення якості звуку.

Взаємодія:

* Отримує команди від модуля керування звуком.
* Передає оброблені аудіо сигнали назад до модуля керування звуком.

4. Модуль роботи з аудіо пристроями

Основні функції:

* Виявлення аудіо: виявляє всі підключені аудіо пристрої.
* Керування аудіо пристроями: виконує команди для налаштування параметрів підключених пристроїв.
* Налаштування параметрів аудіо пристроїв: забезпечує можливість зміни налаштувань підключених аудіо пристроїв.

Взаємодія:

* Отримує команди від модуля керування звуком.
* Повідомляє модуль керування звуком про стан підключених аудіо пристроїв.

Взаємодія між модулями:

* Інтерфейс користувача відправляє команди до модуля керування звуком (наприклад, зміна гучності, застосування фільтра).
* Модуль керування звуком обробляє команди і, залежно від типу команди, передає їх або до модуля обробки сигналів для застосування фільтрів та еквалайзера, або до модуля роботи з аудіо пристроями для налаштування пристроїв.
* Модуль обробки сигналів обробляє аудіо сигнали і повертає їх до модуля керування звуком.
* Модуль роботи з аудіо пристроями взаємодіє з аудіо пристроями і повідомляє про стан пристроїв до модуля керування звуком.
* Модуль керування звуком передає оновлену інформацію до інтерфейсу користувача, щоб відобразити поточний стан звуку та налаштувань.

## 2.5 Вибір засобів для розробки програми

Для розробки програмного забезпечення були проаналізовані наступні інструменти та технології:

1. **NAudio (C#):**
   * ***Огляд:*** Бібліотека для роботи з аудіо в .NET.
   * ***Переваги:*** Потужні можливості для обробки звуку, інтеграція з Windows API, підтримка багатьох аудіо форматів.
   * ***Недоліки:*** Відносно складний у використанні, обмежена документація.
2. **Core Audio API (Windows):**
   * ***Огляд:*** Нативний API для керування звуком в операційній системі Windows.
   * ***Переваги:*** Безпосередній доступ до аудіо пристроїв, висока продуктивність.
   * ***Недоліки:*** Вимагає глибоких знань системного програмування, складність у використанні.

## 2.6 Вибір технології

На основі аналізу було обрано бібліотеку NAudio для розробки програмного забезпечення, оскільки вона забезпечує достатню функціональність для реалізації всіх вимог проекту. Вона також добре інтегрується з .NET платформою, що дозволяє використовувати C# для розробки, що значно спрощує процес розробки та зменшує час на реалізацію проекту.

## Висновок до розділу

Розробка програмного забезпечення для керування звуком є важливим завданням, яке дозволить користувачам покращити якість та зручність управління звуковими параметрами на своїх комп'ютерах. Аналізуючи існуючі рішення та враховуючи переваги та недоліки різних підходів, було розроблено архітектуру, яка забезпечує модульність та ефективність. Використання C# і WinForms дозволить швидко розробити зручний графічний інтерфейс та забезпечити необхідний функціонал для управління звуком.

# РОЗДІЛ 3. Реалізація

## 3.1 Інтерфейс користувача (UI)

Інтерфейс користувача розроблений з використанням WinForms у C#. Він містить такі основні елементи:

* **Головне вікно програми** (Form1):
  + Панель для керування основними налаштуваннями звуку.
  + Список доступних аудіо пристроїв для перемикання.
  + Слайдери для регулювання гучності системи та окремих додатків.
  + Панель налаштувань звукових ефектів та еквалайзера.

## 3.2 Вузли зі структурної схеми

### 3.2.1 Інтерфейс користувача (UI)

Інтерфейс складається з декількох форм та контролів:

* **Form1**: Основне вікно програми.
* **VolumeControl**: Компонент для регулювання гучності.

### 3.2.2 Модуль керування звуком

* **AudioManager**: Основний клас, що взаємодіє з NAudio та Windows API для керування звуком.
  + **AdjustVolume(Device device, float volume)**: Метод для налаштування гучності.
  + **ApplyEffect(Device device, Effect effect)**: Метод для застосування звукових ефектів.

### 3.2.3 Модуль роботи з аудіо пристроями

* **DeviceManager**: Клас, що відповідає за виявлення та керування підключеними аудіо пристроями.
  + **List<AudioDevice> GetConnectedDevices()**: Метод для отримання списку підключених пристроїв.
  + **SelectDevice(AudioDevice device)**: Метод для вибору активного пристрою.

### 3.2.4 Модуль обробки сигналів

* **SignalProcessor**: Клас для обробки аудіо сигналів.
  + **FilterSignal(AudioSignal signal)**: Метод для фільтрації сигналу.
  + **AdjustEqualizer(AudioSignal signal, EqualizerSettings settings)**: Метод для налаштування еквалайзера.

## 3.3 Алгоритми

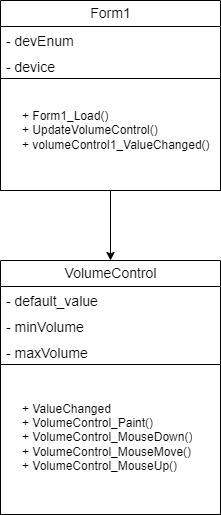
### 3.3.1 Алгоритм регулювання гучності

1. **Вибір пристрою**: Користувач обирає аудіо пристрій через **DeviceSelector**.
2. **Регулювання гучності**: Користувач використовує **VolumeControl** для налаштування рівня гучності.
3. **Застосування змін**: **AudioManager** викликає метод **AdjustVolume()**, який використовує NAudio для зміни гучності обраного пристрою.

### 3.3.2 Алгоритм застосування ефектів

1. **Вибір пристрою**: Користувач обирає аудіо пристрій через **DeviceSelector**.
2. **Налаштування ефектів**: Користувач налаштовує ефекти через **EffectSettings**.
3. **Застосування ефектів**: **AudioManager** викликає метод **ApplyEffect()**, який використовує **SignalProcessor** для обробки сигналу з обраними ефектами.

## 3.4. Діаграма класів



*Рис.3.1.Діаграма класів*

## 3.5 Опис діаграми класів

1. **Form1**: Головне вікно програми, яке містить логіку ініціалізації пристроїв та обробки змін гучності.
2. **VolumeControl**: Клас для управління відображенням і взаємодією з контролом гучності.

Ця реалізація забезпечує модульність та розширюваність системи, що дозволяє легко додавати нові функціональні можливості в майбутньому.

## 3.6 Формат файлів

*Form.cs*  
Цей файл містить основну логіку для керування звуком через об'єкти NAudio, ініціалізацію інтерфейсу та обробку подій.

*VolumeControl.cs*  
Цей файл відповідає за користувацький контроль гучності, включаючи відображення та взаємодію з користувачем через події миші.

*Form1.Designer.cs*  
Цей файл автоматично згенерований дизайнером Windows Forms і містить компонування UI елементів.

*VolumeControl.Designer.cs*  
Цей файл автоматично згенерований дизайнером Windows Forms і містить компонування елементів для користувацького контролю гучності.

*Resources.Designer.cs*  
Цей файл містить ресурсні дані, такі як зображення для контролю гучності.

3.7 Модулі

3.7.1 Модуль керування звуком

Модуль керування звуком забезпечує централізовану обробку всіх команд, пов'язаних з аудіо, включаючи регулювання гучності та застосування ефектів. Він взаємодіє з іншими модулями для забезпечення коректного функціонування системи.

3.7.2 Модуль роботи з аудіо пристроями

Модуль роботи з аудіо пристроями відповідає за виявлення всіх підключених аудіо пристроїв і забезпечує можливість керування ними. Це включає вибір активного пристрою та налаштування його параметрів.  
 3.7.3 Модуль обробки сигналів

Модуль обробки сигналів реалізує алгоритми фільтрації та еквалайзера для поліпшення якості звуку. Він отримує аудіо сигнали від модуля керування звуком, обробляє їх і повертає оброблені сигнали назад.

3.7.4 Взаємодія між модулями

Взаємодія між модулями забезпечує цілісність і узгодженість роботи системи. Кожен модуль має чітко визначені функції та відповідальність, що дозволяє забезпечити ефективне керування звуком і високу якість обробки аудіо сигналів.

3.8 Користувацький досвід (UX)

Основна мета інтерфейсу користувача — забезпечити інтуїтивно зрозуміле та зручне управління звуком. Важливими аспектами є:

* Простота використання: Всі основні функції доступні з головного вікна.
* Візуальна привабливість: Інтерфейс має сучасний дизайн з чіткими та зрозумілими елементами управління.
* Відповідність очікуванням користувача: Всі налаштування та функції розташовані так, як це звично користувачам інших аудіо програм.

3.9 Тестування

Для забезпечення якості програмного забезпечення передбачено кілька рівнів тестування:

* Юніт-тестування: Кожен модуль має бути протестований окремо для перевірки коректності його роботи.
* Інтеграційне тестування: Взаємодія між модулями має бути протестована для перевірки коректності спільної роботи.
* Тестування користувацького інтерфейсу: Перевірка зручності використання та відсутності багів у UI.

## 3.10 Документація

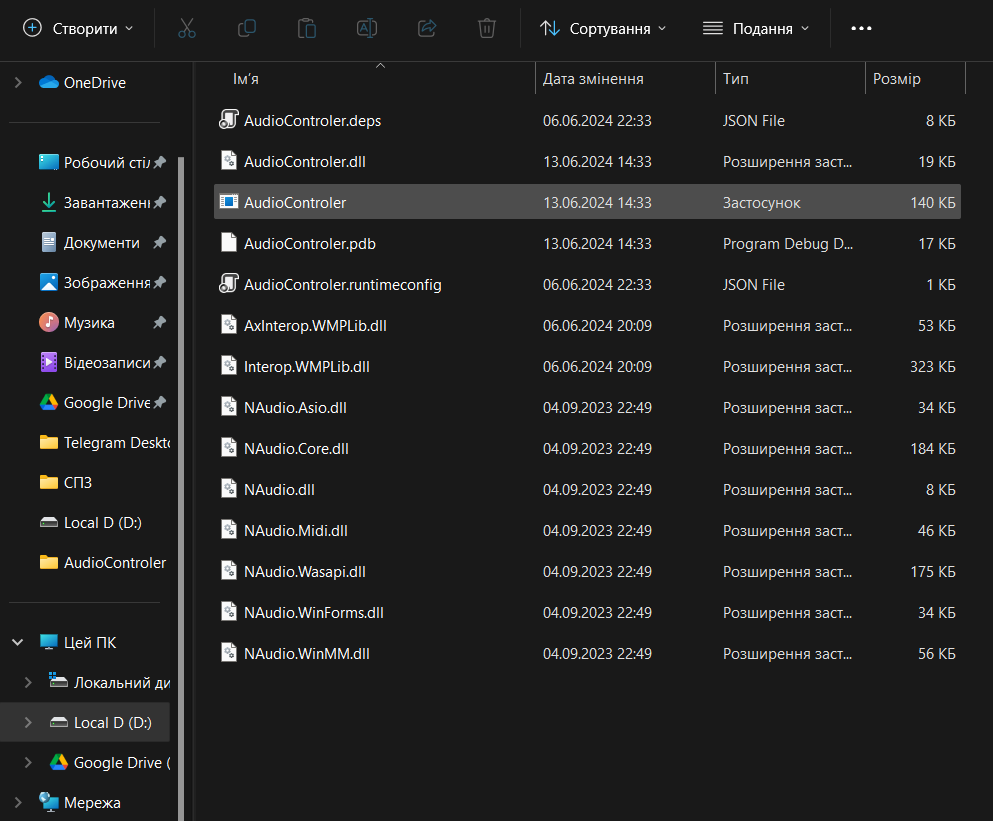
Документація програмного забезпечення включає:

* Технічну документацію: Опис архітектури, алгоритмів та класів.
* Користувацьку документацію: Інструкції для користувачів щодо використання програмного забезпечення.
* Документацію для розробників: Керівництво для розробників щодо розширення та підтримки коду.

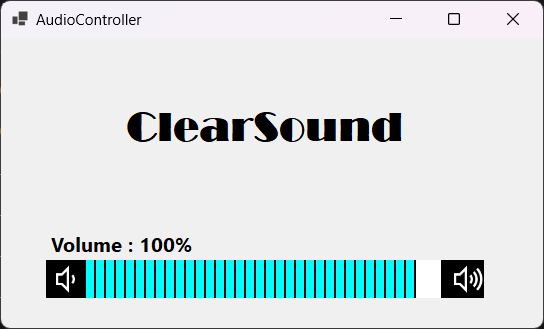
Висновок до розділу

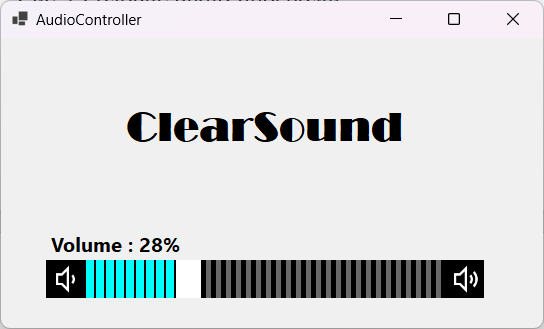
Ця програма надає користувачам зручний інтерфейс для керування гучністю та застосування звукових ефектів, забезпечуючи при цьому гнучкість та розширюваність для майбутніх покращень та додаткового функціоналу.

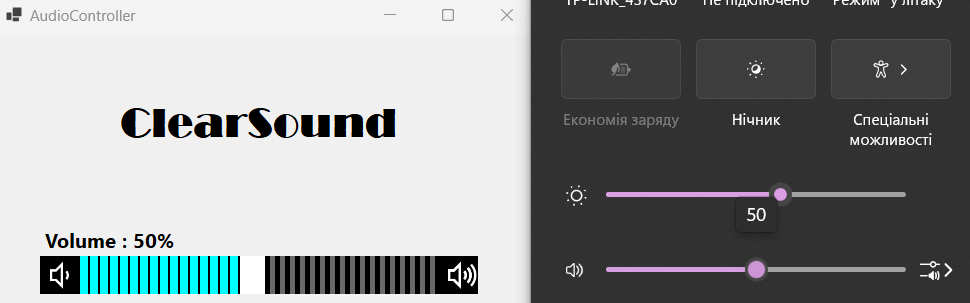
# РОЗДІЛ 4. Тестування

Керування звуком відбувається безпосередньо в системі, тобто іде відтворення гучності саме на теперішній діючий девайс і при зміні гучності в програмі, мікшер гучності Windows, теж змінює своє значення, а також змінна ставиться така, яка була до запуску програми.  
  
  
  
 

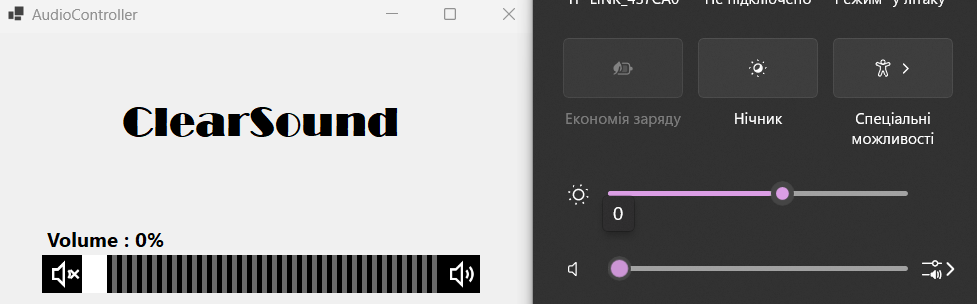
*Рис.4.1.Запуск програми через .exe*

*  
  
 Рис.4.2.Головне вікно програми*

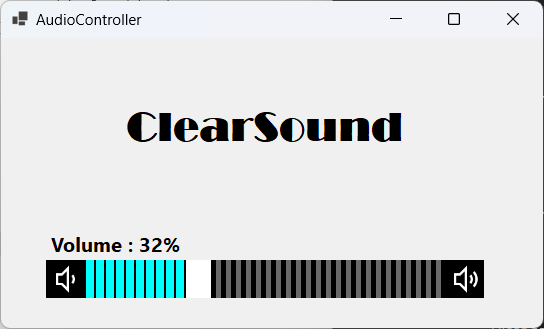
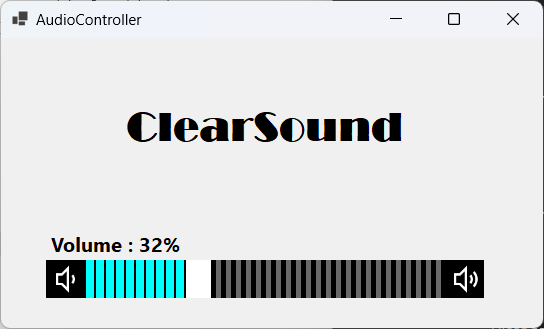
*  
  
 Рис.4.3.Приклад зміни гучності*

**

*Рис.4.4.Синхронізація з мікшером гучності Windows(Приклад 1)*

****

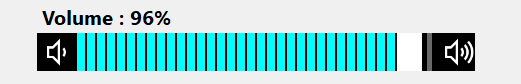
*Рис.4.5.Синхронізація з мікшером гучності Windows(Приклад 2)*

**

*Рис.4.6.Приклад збереження змін(зліва до закритя прогмами, справа після повторного відкритя)*



*Рис.4.7.Вид іконок динаміка при 0%*

****

*Рис.4.8.Вид іконок динаміка при >50%*

Висновок до розділу

Після проведення тестування, можна зробити такі висновки, що програма повністю синхронізується із системою Windows. Наявний функціонал збереження змін. Динамічні зображення, які реагують на зміну гучності, а також інтуєтивно зрозумілий інтерфейс.

# РОЗДІЛ 5. ВИСНОВКИ

## 5.1 Розроблений додаток:

У данному курсовому проекті була створена програма для керування аудіопристроями, яка дозволяє користувачам налаштовувати гучність звуку на комп'ютері за допомогою зручного графічного інтерфейсу.

## 5.2 Основні функції:

1. **Керування гучністю**:
   * Можливість регулювання гучності за допомогою слайдера.
   * Відображення поточного рівня гучності у відсотках.
2. **Вибір аудіопристрою**:
   * Вибір та налаштування поточного аудіопристрою (динаміки, навушники тощо).
3. **Зберігання налаштувань**:
   * Збереження користувацьких налаштувань при закритті програми і відновлення їх при повторному запуску.

## 5.3 Виконання завдання:

Дана програма може в собі підтримувати ще більший функціонал, наприклад керування звуку мікрофона, або випадаюче меню із запропонованими пристроями. Наразі поточна програма успішно реалізує керування звуком і зокрема синхронізацію з системою Windows. Тестування підтвердило, що всі функції працюють коректно і відповідають визначеним вимогам.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* + 1. C# docs - get started, tutorials, reference. URL: <https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp> (Дата звернення 20.05.24).
    2. GitHub - vuplea/CoreAudioAPI: Provides a lightweight set. URL: <https://github.com/vuplea/CoreAudioAPI> (Дата звернення 14.05.24).
    3. NAudio.Core. URL: <https://www.nuget.org/packages/NAudio.Core#readme-body-tab> (Дата звернення 01.06.24)
    4. Windows Forms .NET. URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/controls/layout?view=netdesktop-8.0>(Дата звернення 02.06.24)
    5. Windows Forms. URL: <https://ru.wikipedia.org/wiki/Windows_Forms> (Дата звернення 03.06.24)
    6. Відомості про API аудіо в Windows Core - Win32 apps. URL: <https://learn.microsoft.com/ru-ru/windows/win32/coreaudio/about-the-windows-core-audio-apis> (Дата звернення 29.05.24)

# ДОДАТОК А

*Form1.cs*

using System;

using System.Windows.Forms;

using NAudio.CoreAudioApi;

namespace AudioControler

{

public partial class Form1 : Form

{

private MMDeviceEnumerator devEnum;

private MMDevice device;

public Form1()

{

InitializeComponent();

devEnum = new MMDeviceEnumerator();

}

private void Form1\_Load(object sender, EventArgs e)

{

try

{

device = devEnum.GetDefaultAudioEndpoint(DataFlow.Render, Role.Multimedia);

volumeControl1.Value = (int)(device.AudioEndpointVolume.MasterVolumeLevelScalar \* 100);

volumeControl1.ValueChanged += volumeControl1\_ValueChanged;

}

catch (Exception ex)

{

MessageBox.Show("Error initializing audio device: " + ex.Message);

}

}

private void AudioEndpointVolume\_OnVolumeNotification(AudioVolumeNotificationData data)

{

if (InvokeRequired)

{

Invoke(new Action(() => UpdateVolumeControl(data.MasterVolume)));

}

else

{

UpdateVolumeControl(data.MasterVolume);

}

}

private void UpdateVolumeControl(float volume)

{

if (volumeControl1 != null && !volumeControl1.IsDisposed)

{

volumeControl1.Value = (int)(volume \* 100);

}

}

private void volumeControl1\_ValueChanged(object sender, EventArgs e)

{

if (device != null)

{

device.AudioEndpointVolume.MasterVolumeLevelScalar = volumeControl1.Value / 100.0f;

lbl\_volume.Text = "Volume : " + volumeControl1.Value.ToString() + "%";

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

lbl\_volume.Text = "Volume : " + volumeControl1.Value.ToString() + "%";

}

}

}

*Form1.Designer.cs*

namespace AudioControler

{

partial class Form1

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing)

{

if (device != null)

{

device.AudioEndpointVolume.OnVolumeNotification -= AudioEndpointVolume\_OnVolumeNotification;

device.Dispose();

device = null;

}

if (devEnum != null)

{

devEnum.Dispose();

devEnum = null;

}

if (components != null)

{

components.Dispose();

}

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

private void InitializeComponent()

{

components = new System.ComponentModel.Container();

volumeControl1 = new VolumeControl();

lbl\_volume = new Label();

timer1 = new System.Windows.Forms.Timer(components);

prg\_Name = new Label();

SuspendLayout();

//

// volumeControl1

//

volumeControl1.BackColor = Color.Black;

volumeControl1.Bar\_color = Color.Aqua;

volumeControl1.Location = new Point(45, 222);

volumeControl1.Max = 100;

volumeControl1.Min = 0;

volumeControl1.Name = "volumeControl1";

volumeControl1.Size = new Size(438, 38);

volumeControl1.TabIndex = 0;

volumeControl1.Value = 40;

volumeControl1.ValueChanged += volumeControl1\_ValueChanged;

//

// lbl\_volume

//

lbl\_volume.AutoSize = true;

lbl\_volume.Font = new Font("Segoe UI", 12F, FontStyle.Bold, GraphicsUnit.Point, 204);

lbl\_volume.Location = new Point(45, 191);

lbl\_volume.Name = "lbl\_volume";

lbl\_volume.Size = new Size(135, 28);

lbl\_volume.TabIndex = 1;

lbl\_volume.Text = "Volume: 40%";

//

// timer1

//

timer1.Enabled = true;

timer1.Tick += timer1\_Tick;

//

// prg\_Name

//

prg\_Name.AutoSize = true;

prg\_Name.Font = new Font("Broadway", 25.2F, FontStyle.Bold, GraphicsUnit.Point, 0);

prg\_Name.Location = new Point(117, 65);

prg\_Name.Name = "prg\_Name";

prg\_Name.Size = new Size(297, 47);

prg\_Name.TabIndex = 2;

prg\_Name.Text = "ClearSound";

prg\_Name.Click += prg\_Name\_Click;

//

// Form1

//

AutoScaleDimensions = new SizeF(8F, 20F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(542, 290);

Controls.Add(prg\_Name);

Controls.Add(lbl\_volume);

Controls.Add(volumeControl1);

Name = "Form1";

StartPosition = FormStartPosition.CenterScreen;

Text = "AudioController";

Load += Form1\_Load;

ResumeLayout(false);

PerformLayout();

}

private void prg\_Name\_Click(object sender, EventArgs e)

{

throw new NotImplementedException();

}

#endregion

private VolumeControl volumeControl1;

private Label lbl\_volume;

private System.Windows.Forms.Timer timer1;

private Label prg\_Name;

}

}

*VolumeControl.cs*

using System;

using System.Drawing;

using System.Windows.Forms;

namespace AudioControler

{

public partial class VolumeControl : UserControl

{

public event EventHandler ValueChanged;

public VolumeControl()

{

InitializeComponent();

this.Size = new Size(350, 30);

this.BackColor = Color.Black;

DoubleBuffered = true;

}

int default\_value = 40;

int minVolume = 0, maxVolume = 100;

public int Max { get { return maxVolume; } set { maxVolume = value; Invalidate(); } }

public int Min { get { return minVolume; } set { minVolume = value; Invalidate(); } }

public int Value

{

get { return default\_value; }

set

{

if (default\_value != value)

{

default\_value = value;

Invalidate();

OnValueChanged(EventArgs.Empty);

}

}

}

public int gap = 10;

Color b\_color = Color.Aqua;

public Color Bar\_color { get { return b\_color; } set { b\_color = value; Invalidate(); } }

private void VolumeControl\_Paint(object sender, PaintEventArgs e)

{

int start\_point = 40;

SolidBrush sb = new SolidBrush(Color.DimGray);

for (int j = 0; j < (Max \* ClientSize.Width / Max - 75) / gap; j++)

{

e.Graphics.FillRectangle(sb, new Rectangle(start\_point, 0, gap - 5, ClientSize.Height));

start\_point += gap;

}

int buffer\_point = 40;

SolidBrush br = new SolidBrush(b\_color);

for (int i = 0; i < (default\_value \* ClientSize.Width / Max - default\_value) / gap; i++)

{

e.Graphics.FillRectangle(br, new Rectangle(buffer\_point, 0, gap - 2, ClientSize.Height));

buffer\_point += gap;

}

int thumb\_size = 25;

SolidBrush thumb = new SolidBrush(Color.White);

e.Graphics.FillRectangle(thumb, new Rectangle(buffer\_point, 0, thumb\_size, ClientSize.Height));

if (default\_value >= Min)

{

Image left\_img = Properties.Resources.down\_img;

e.Graphics.DrawImage(left\_img, 5, 0, ClientSize.Height, ClientSize.Height);

}

if (default\_value <= 50)

{

Image right\_img = Properties.Resources.mid\_img;

e.Graphics.DrawImage(right\_img, ClientSize.Width - 35, 0, ClientSize.Height, ClientSize.Height);

}

if (default\_value <= Min)

{

Image left\_img = Properties.Resources.mute\_img;

e.Graphics.DrawImage(left\_img, 5, 0, ClientSize.Height, ClientSize.Height);

}

if (default\_value >= 50)

{

Image right\_img = Properties.Resources.high\_img;

e.Graphics.DrawImage(right\_img, ClientSize.Width - 35, 0, ClientSize.Height, ClientSize.Height);

}

}

private void Bar\_value(float value)

{

if (value < Min) value = Min;

if (value > Max) value = Max;

if (default\_value == value) return;

default\_value = (int)value;

this.Refresh();

OnValueChanged(EventArgs.Empty);

}

private float thumb\_value(int x)

{

return Min + (Max - Min) \* x / (float)(ClientSize.Width);

}

bool mouse = false;

private void VolumeControl\_MouseDown(object sender, MouseEventArgs e)

{

mouse = true;

Bar\_value(thumb\_value(e.X));

}

private void VolumeControl\_MouseMove(object sender, MouseEventArgs e)

{

if (!mouse) return;

Bar\_value(thumb\_value(e.X));

}

private void VolumeControl\_MouseUp(object sender, MouseEventArgs e)

{

mouse = false;

OnValueChanged(EventArgs.Empty);

}

protected virtual void OnValueChanged(EventArgs e)

{

ValueChanged?.Invoke(this, e);

}

}

}

*VolumeControl.Designer.cs*

namespace AudioControler

{

partial class VolumeControl

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Component Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.SuspendLayout();

//

// VolumeControl

//

this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 20F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.Name = "VolumeControl";

this.Size = new System.Drawing.Size(350, 30);

this.Paint += new System.Windows.Forms.PaintEventHandler(this.VolumeControl\_Paint);

this.MouseDown += new System.Windows.Forms.MouseEventHandler(this.VolumeControl\_MouseDown);

this.MouseMove += new System.Windows.Forms.MouseEventHandler(this.VolumeControl\_MouseMove);

this.MouseUp += new System.Windows.Forms.MouseEventHandler(this.VolumeControl\_MouseUp);

this.ResumeLayout(false);

}

#endregion

}

}

*Resources.Designer.cs*namespace AudioControler.Properties

{

using System;

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBuilder", "4.0.0.0")]

[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]

[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

internal class Resources

{

private static global::System.Resources.ResourceManager resourceMan;

private static global::System.Globalization.CultureInfo resourceCulture;

[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1811:AvoidUncalledPrivateCode")]

internal Resources()

{

}

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]

internal static global::System.Resources.ResourceManager ResourceManager

{

get

{

if ((resourceMan == null))

{

global::System.Resources.ResourceManager temp = new global::System.Resources.ResourceManager("AudioControler.Properties.Resources", typeof(Resources).Assembly);

resourceMan = temp;

}

return resourceMan;

}

}

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]

internal static global::System.Globalization.CultureInfo Culture

{

get

{

return resourceCulture;

}

set

{

resourceCulture = value;

}

}

internal static System.Drawing.Bitmap down\_img

{

get

{

object obj = ResourceManager.GetObject("down\_img", resourceCulture);

return ((System.Drawing.Bitmap)(obj));

}

}

internal static System.Drawing.Bitmap high\_img

{

get

{

object obj = ResourceManager.GetObject("high\_img", resourceCulture);

return ((System.Drawing.Bitmap)(obj));

}

}

internal static System.Drawing.Bitmap mid\_img

{

get

{

object obj = ResourceManager.GetObject("mid\_img", resourceCulture);

return ((System.Drawing.Bitmap)(obj));

}

}

internal static System.Drawing.Bitmap mute\_img

{

get

{

object obj = ResourceManager.GetObject("mute\_img", resourceCulture);

return ((System.Drawing.Bitmap)(obj));

}

}

}

}