**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інженерії програмного забезпечення**

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно - орієнтоване програмування»

на тему: **MP3-програвач на мові Python**

121 – КР.ПЗ.00 – 209.1910901

**Виконав: студент 2 курсу, групи 209**

**спеціальності 121 «Інженерія програмного забезпечення»**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К. О. Бєктін

(підпис, ініціали та прізвище)

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

(дата)

**Керівник: ст. викладач**

**Боровльова С.Ю.**

Національна   шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_ \_\_\_\_\_\_\_\_

**м. Миколаїв – 2021 рік**

**ЗМІСТ**

[ВСТУП 3](#_Toc71106034)

[ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕйСУ 4](#_Toc71106035)

[Створення дизайну головного вікна програми 4](#_Toc71106036)

[Створення дизайну вікна «About» 6](#_Toc71106037)

[Створення логотипу 7](#_Toc71106038)

[ПРОГРАМНА РЕАЛІЗАЦІЯ 9](#_Toc71106039)

[Опис можливостей управління списком файлів 9](#_Toc71106040)

[Опис механізму програвання аудіо 12](#_Toc71106041)

[Опис механізму роботи з метаданими 15](#_Toc71106042)

[Реалізація інтерфейсу 17](#_Toc71106043)

[Структура головного вікна 17](#_Toc71106044)

[тестування готового застосунку 21](#_Toc71106045)

[Маніпуляції зі списком файлів 21](#_Toc71106046)

[Програвання аудіо 26](#_Toc71106047)

[висновки 31](#_Toc71106048)

[список використаних джерел 32](#_Toc71106049)

# ВСТУП

Настільні застосунки являють собою найрізноманітнішу галузь програмних продуктів. Саме у цій сфері програміста чекають завдання, що потребують не тільки гарної кодової бази, а й дуже розвинену креативність та уміння винаходити. Програми для 3D-моделювання, роботи із математичними обчисленнями, IDE чи DAW(Digital Audio Workstation) – усе це є дуже складними типами застосунків, потребуючими надзвичайно глибоких знань з галузі використання(користувача) та здатність переносити ці концепції у зовсім інше оточення - код.

В ході цієї курсової роботи буде створено MP3-програвач з використанням можливостей мови Python. Його назва – «Mini MP3-Player», або просто «MMP3». Обрана мова програмування славиться можливістю швидкої розробки програмних рішень. Ця особливість компенсується швидкістю роботи програми, але у рамках цього проєкту це не є критичним.

Завданням курсової роботи є створення «легкого», зручного у використанні MP3-програвача з інтуїтивно зрозумілим інтерфейсом.

Мета роботи – побудова власного настільного GUI-застосунку з можливістю подальшої модернізації та особистого використання.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Спроєктувати користувацький інтерфейс
2. Створити логотип
3. Визначити ключові сутності у коді
4. Побудувати користувацький інтерфейс
5. Надати елементам інтерфейсу функціонал
6. Протестувати готовий продукт

Курсова робота складається з 32 сторінок, містить 22 ілюстрації, 10 використаних джерел.

# ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕйСУ

Практика показує, що половина успіху програмного продукту залежить від його оформлення. Дуже малий відсоток людей хотів би користуватись застосунком, графічний інтерфейс якого був би погано спроєктований. Також, не можна забувати про естетику. Якщо поглянути на дизайн сучасних програмних рішень, то можна одразу побачити, що кожна компанія має при собі великий підрозділ присвячений саме цьому.

Дизайн даного застосунку було вирішено реалізувати у стилі «мінімалізм» без використання складної графіки.

## Створення дизайну головного вікна програми

Спочатку зобразимо верхню частину вікна. У ній має розміщатися зображення обкладинки аудіо-файлу, його назва, інформація про час та елементи керування самим програвачем: кнопки, що відповідають за програвання та його режими, слайдери позиції у аудіо-файлі та гучності.

Колірна гама представлена наступним набором:

Задній фон основного вікна та фон обраних файлів у списку - #2B2B2B

Задній фон списку аудіо-файлів - #3C3F41

Текст та елементи керування - #BBBBBB

Текст та елементи керування (при наведенні) - #F0F0F0

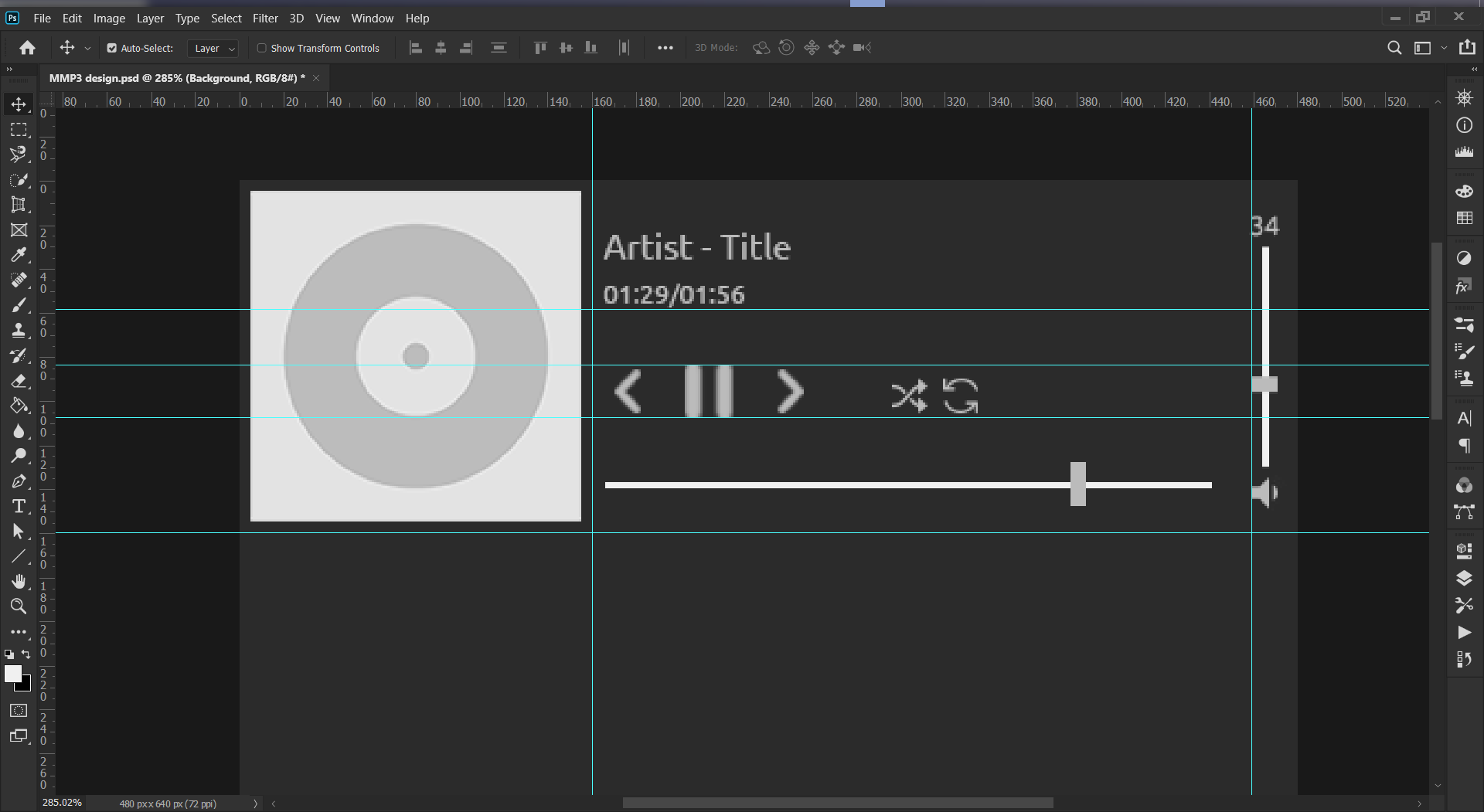


Рисунок 1 – Дизайн верхньої частини вікна застосунку

Наступним кроком є додавання нижньої групи управління списком файлів.

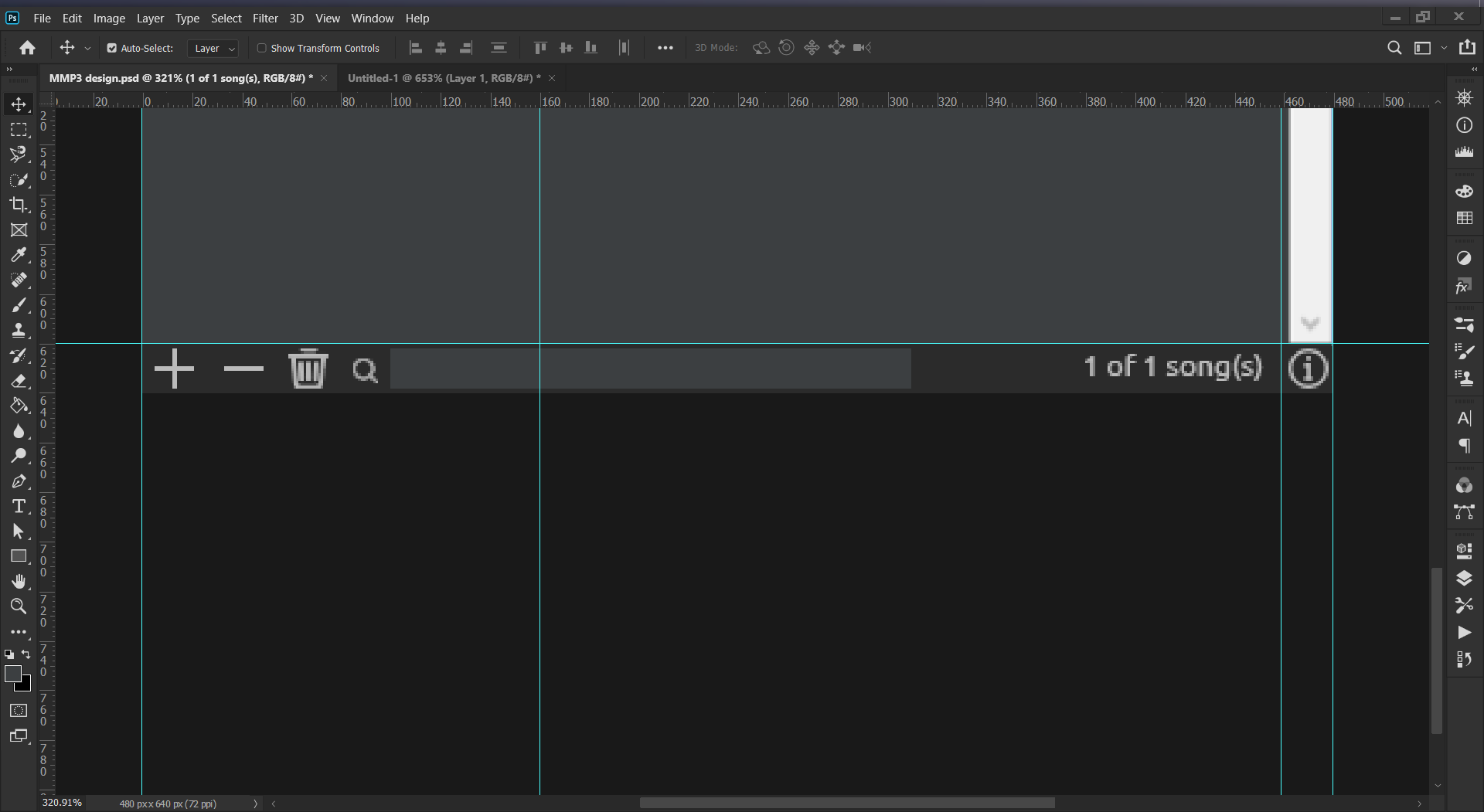


Рисунок 2 – Дизайн панелі керування списком файлів

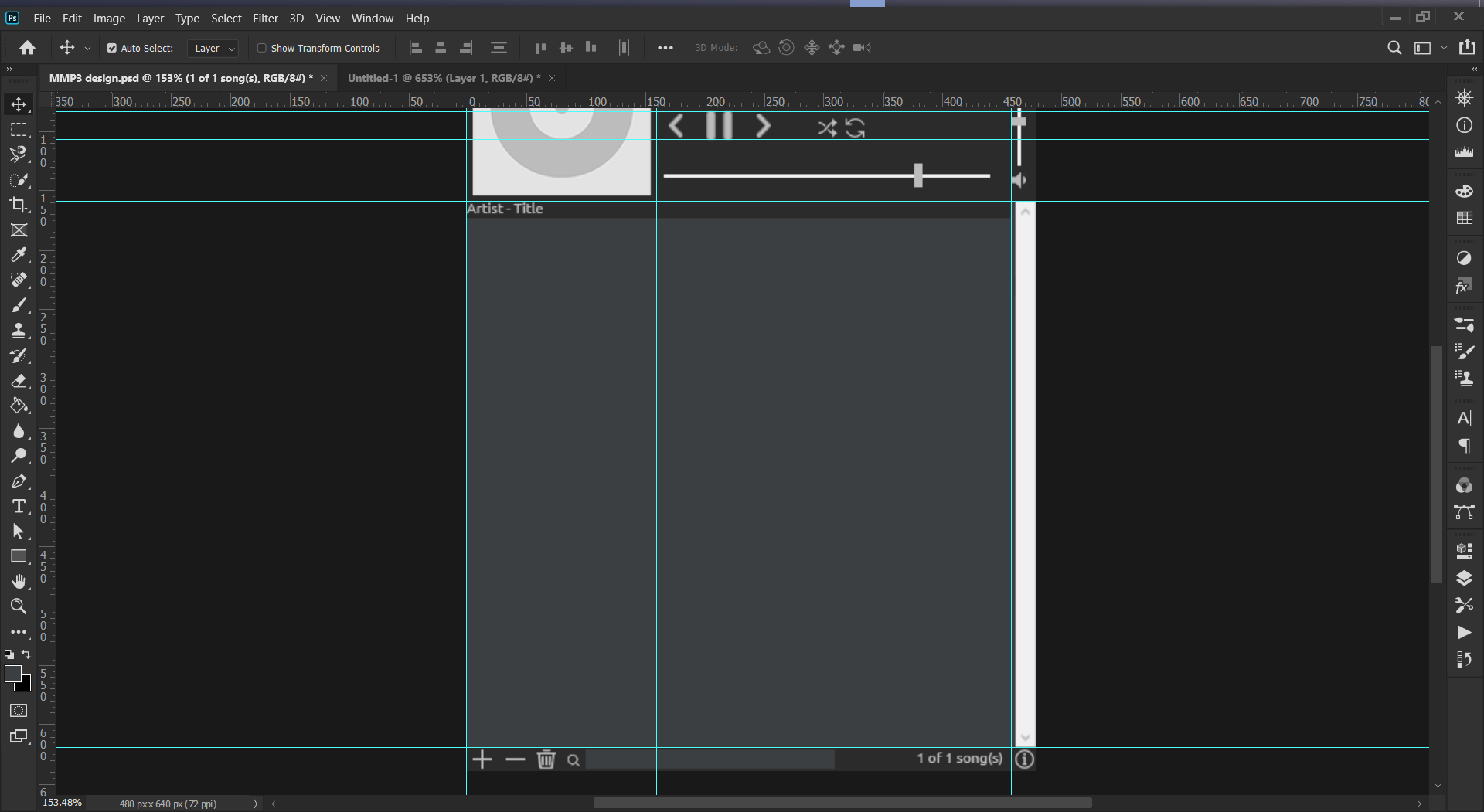


Рисунок 3 – Фрейм для списку файлів

## Створення дизайну вікна «About»

Дане вікно зберігає мінімум важливої інформації про застосунок та посилання для зворотного зв’язку і не тільки.

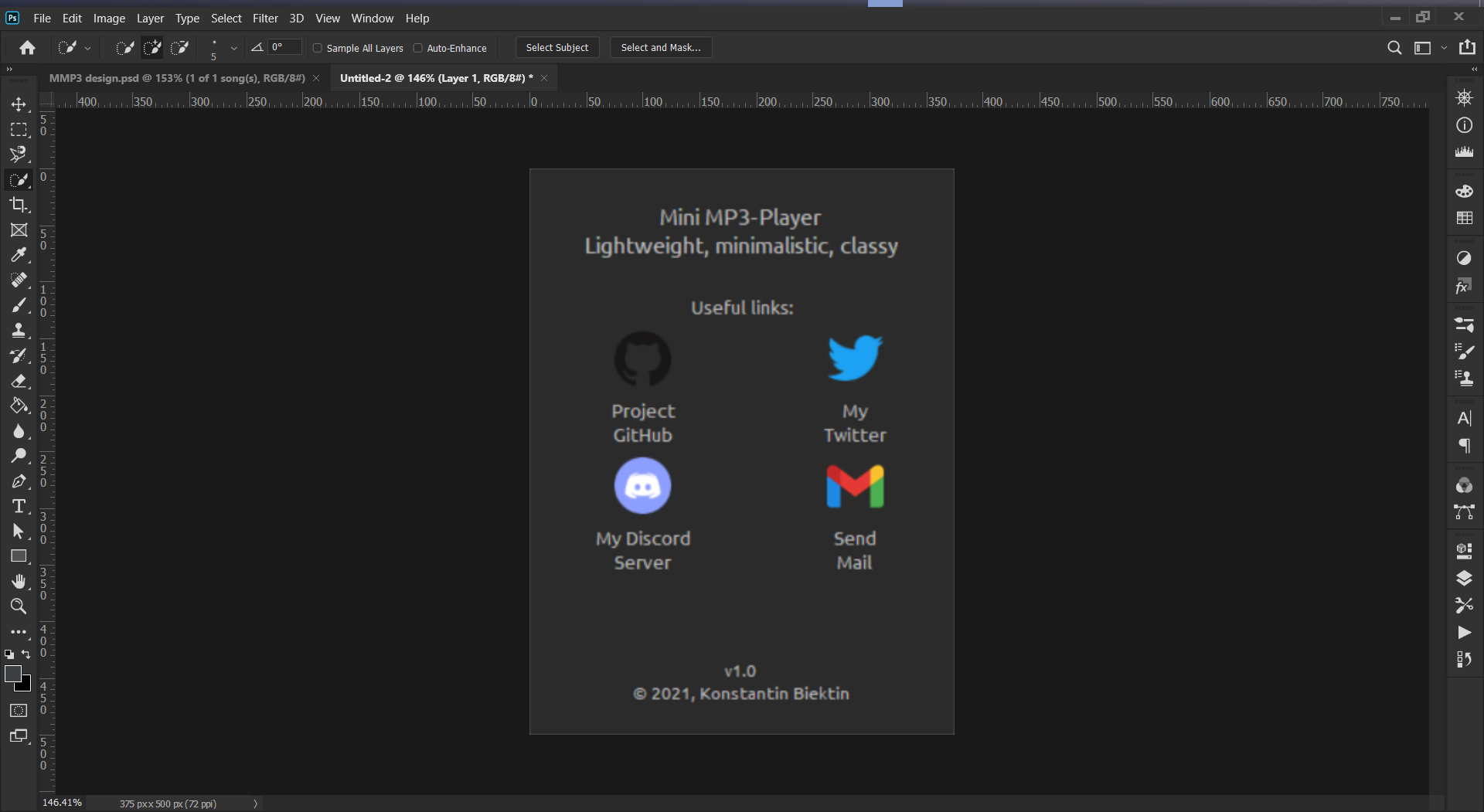


Рисунок 4 – Дизайн вікна «About»

Іконки є кнопками, що відкривають браузер та відповідне посилання.

## Створення логотипу

За основу логотипу було вирішено взяти образ кнопки «play». Програмним забезпеченням було обрано Adobe Photoshop 2018 CC.

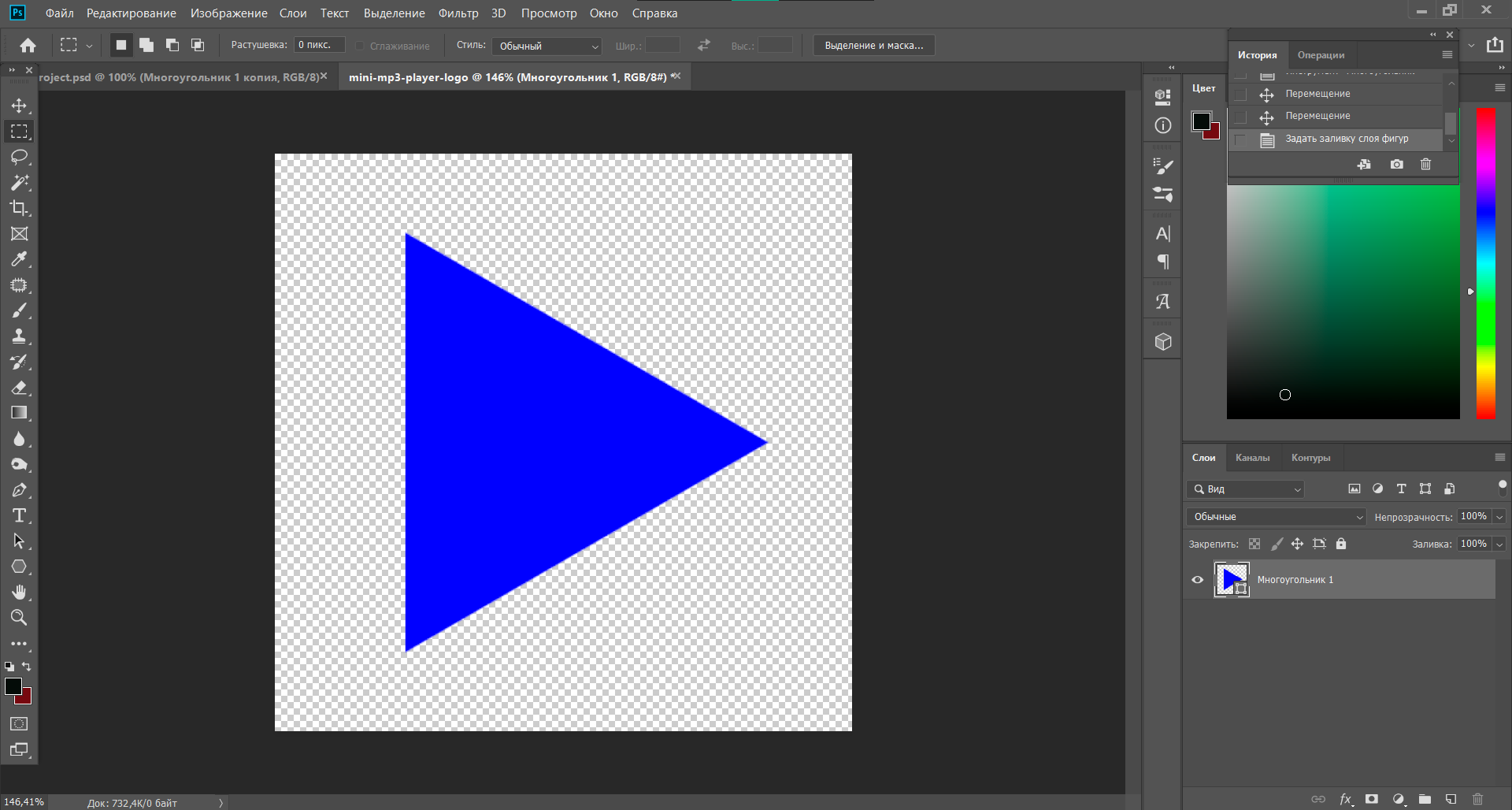


Рисунок 5 – Початок роботи

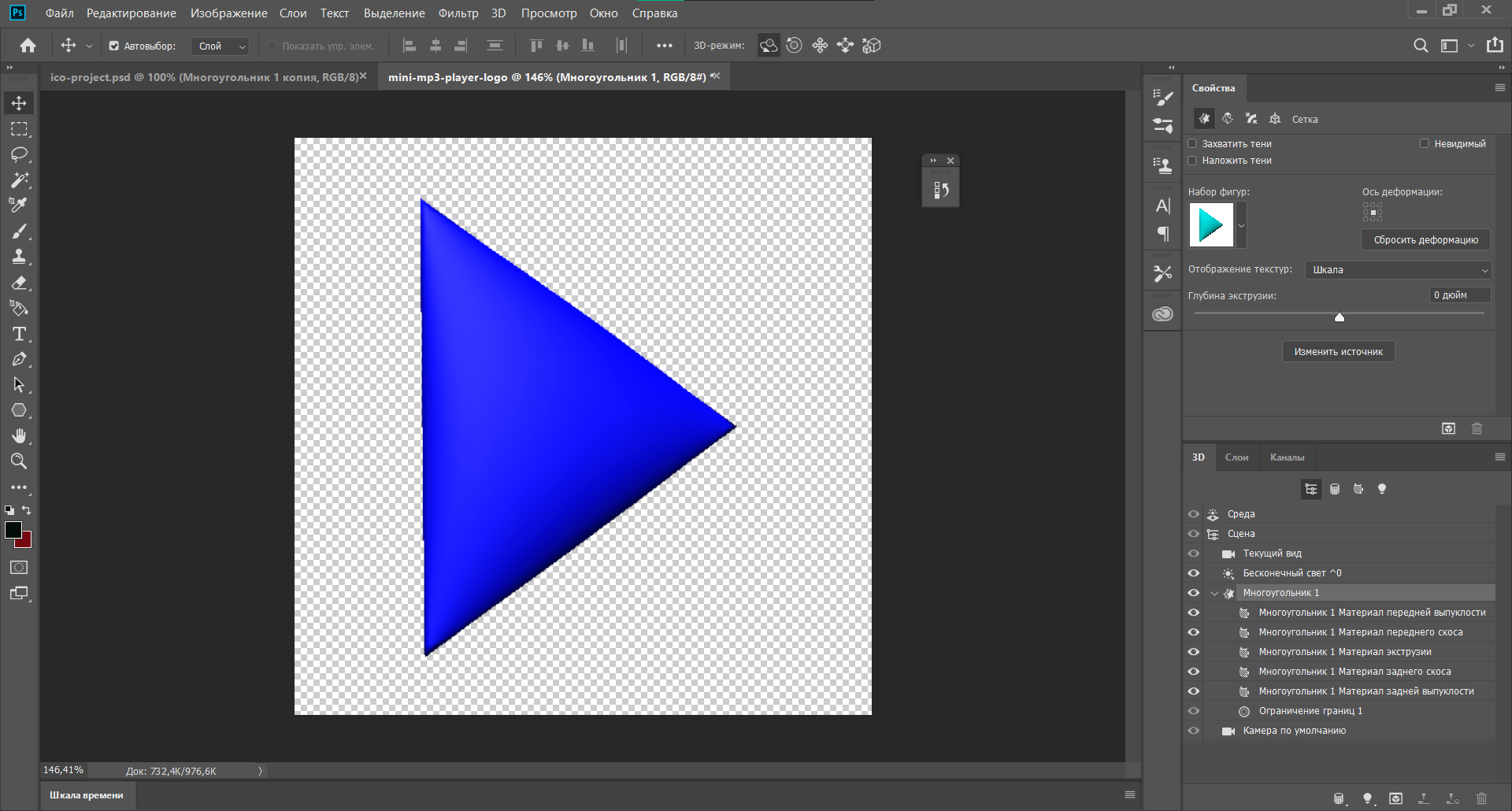


Рисунок 6 – Переведення об’єкта у 3D.

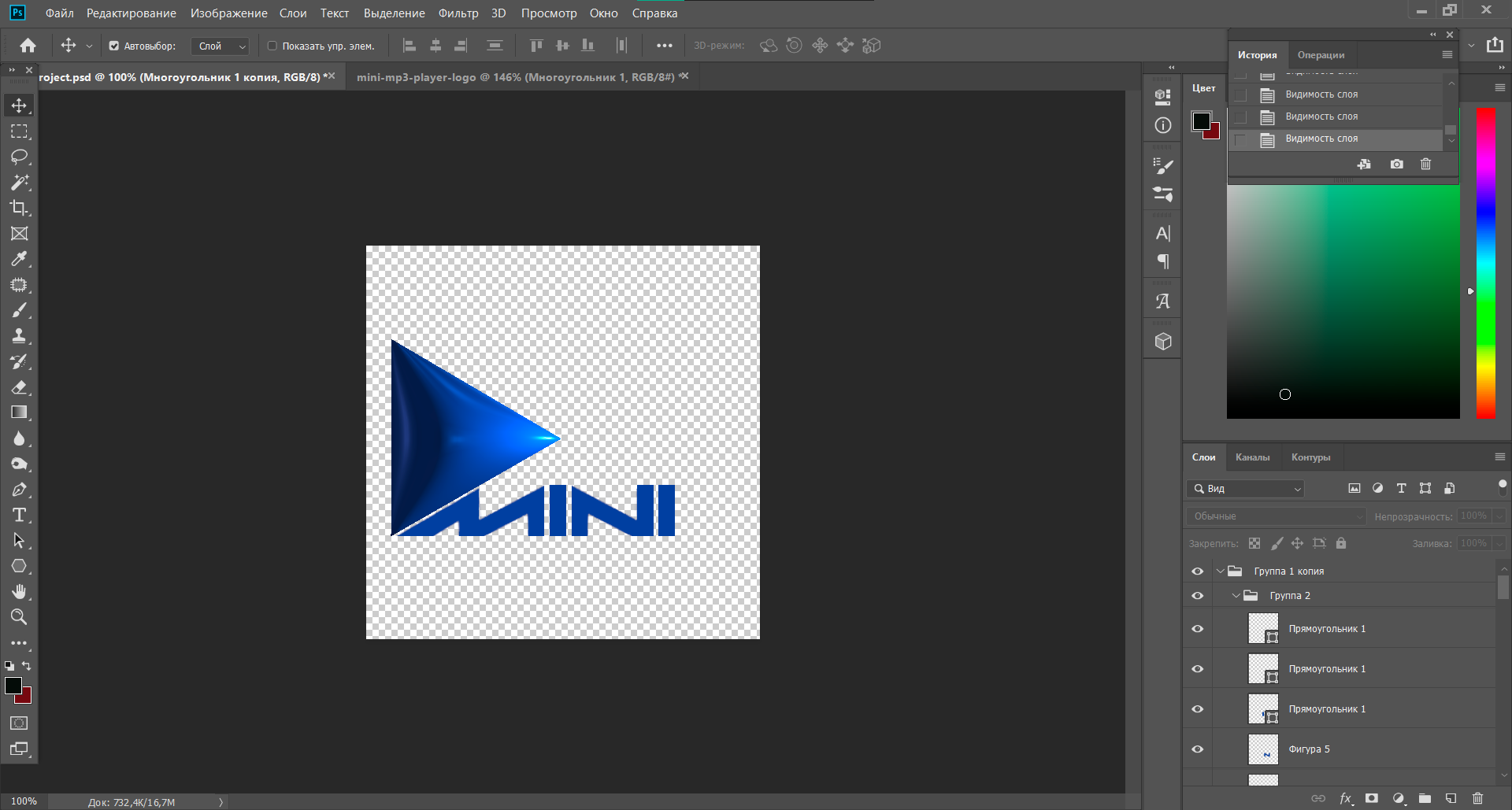


Рисунок 7 – Додавання ефектів та підпису



Рисунок 8 – Допрацювання отриманого зображення



Рисунок 9 – Додавання фону

# ПРОГРАМНА РЕАЛІЗАЦІЯ

Реалізація користувацького інтерфейсу за допомогою модуля Tkinter базується на створенні вікон, у які поміщаються фрейми, наповнені іншими віджетами, які також можуть бути обгорнуті у фрейми.

## Опис можливостей управління списком файлів

Проаналізувавши список класів модуля Tkinter, для представлення списку завантажених файлів було вирішено використати Listbox.

self.songsList = tk.Listbox(self.songsListFrame, yscrollcommand=self.songsListScrollbar.set,

bg=GUI.songsListBackgroundColor,

fg=GUI.foregroundColor, selectbackground=GUI.backgroundColor,

borderwidth=0,

highlightthickness=0)

Для того, аби почати програвати файл, необхідно його додати до списку. За це, і не тільки, відповідає створений клас SongsBox. Для цього на панелі управління списком файлів присутня кнопка зі знаком «+». Після її натиснення відкривається вікно для вибору файлів з файлової системи із обмеженням на формат «.mp3». Для збереження шляхів до файлів створено список.

self.songsFullPaths = []

Для видалення файлів зі списку необхідно натиснути сусідню кнопку зі знаком «-». Після цього обрати зі списку ті файли, які б ви хотіли прибрати та натиснути на кнопку зі зображенням сміттєвого баку.

Метод для увімкнення режиму видалення зі списку:

**def** song\_removing\_mode\_switch(self):

**if** **not** self.isSongRemovingMode:

self.isSongRemovingMode = True

self.gui.songsList.config(selectmode=tk.EXTENDED)

self.gui.songsList.unbind("<ButtonRelease-1>")

self.gui.btn\_song\_removing\_mode\_switch.config(bg="#6E6E6E")

self.gui.btn\_remove\_selected\_songs.config(state=tk.NORMAL)

**else**:

self.isSongRemovingMode = False

self.gui.songsList.config(selectmode=tk.BROWSE)

self.\_\_bind\_songs\_box\_event()

  self.gui.btn\_song\_removing\_mode\_switch.config(bg=GUI.GUI.backgroundColor)

self.gui.btn\_remove\_selected\_songs.config(state=tk.DISABLED)

self.select\_current\_index()

Та для видалення обраних елементів:

**def** \_\_remove\_selected\_songs(self):

selected\_songs\_indices = self.gui.songsList.curselection()

selected\_songs\_titles = []

songs\_full\_paths\_list = list(self.songsFullPaths)

all\_songs\_titles = list(self.gui.songsList.get(0, tk.END))

**for** i **in** selected\_songs\_indices:

selected\_songs\_titles.append(self.gui.songsList.get(i))

**for** song\_title **in** selected\_songs\_titles:

**for** song\_full\_path **in** songs\_full\_paths\_list:

**if** song\_title **in** song\_full\_path:

songs\_full\_paths\_list.remove(song\_full\_path)

**if** self.currentSongFullPath.find(song\_title) != -1:

self.reset\_player\_state()

self.songsFullPaths = tuple(songs\_full\_paths\_list)

self.songsListSize = len(self.songsFullPaths)

**if** self.songsListSize == 0:

self.currentSongIndex = -1

**for** song\_title **in** selected\_songs\_titles:

all\_songs\_titles.remove(song\_title)

self.gui.songsList.delete(0, tk.END)

self.gui.songsList.insert(tk.END, \*all\_songs\_titles)

self.\_\_find\_song\_by\_name()

self.select\_current\_index()

self.update\_song\_qty\_label()

Також, можна здійснювати пошук по списку. Для цього передбачено поле для вводу. Пошук залежить від регістру уведеного тексту. Визначена спеціальна змінна типу StringVar(), яка служить контейнером для вмісту текстового поля, який використовується пошуковим механізмом.

self.query = tk.StringVar()

Ще однією з можливих операцій над списком є перемішування. Варто відмітити, що якщо під час виконання даної функції грала пісня, то аби не загубити її, вона завжди буде першою у списку.

**def** shuffle\_songs(self):

songs\_full\_paths\_list = list(self.songsFullPaths)

random.shuffle(songs\_full\_paths\_list)

**if** self.playbackControls.isSongLoaded:

songs\_full\_paths\_list.remove(self.currentSongFullPath)

songs\_full\_paths\_list.insert(0, self.currentSongFullPath)

self.currentSongIndex = 0

self.songsFullPaths = songs\_full\_paths\_list

songs\_names\_list = [name[name.rfind("/") + 1:-4] **for** name **in** songs\_full\_paths\_list]

self.gui.songsList.delete(0, tk.END)

self.gui.songsList.insert(tk.END, \*songs\_names\_list)

self.select\_current\_index()

self.update\_song\_qty\_label()

## Опис механізму програвання аудіо

За аудіо-складову відповідає модуль pygame.mixer.music з набору модулів «PyGame», що використовується для створення відеоігор.

Для того, щоб запустити пісню, треба просто обрати її лівим кліком миші зі списку. Це можливо завдяки слухачу подій для Listbox-об’єкта:

**def** \_\_bind\_songs\_box\_event(self):

self.gui.songsList.bind("<ButtonRelease-1>", self.\_\_play\_selected\_song)

Після кліку відбувається ланцюг викликів методів, ключовим з яких є один з класу PlaybackControls:

**def** play\_selected\_song(self):

self.gui.btn\_play.config(state=tk.NORMAL)

self.\_\_cycle\_wait\_for\_song\_to\_end()

self.isSongPaused = False

self.\_\_change\_play\_button\_to\_pause()

pygame.mixer.music.load(self.songsBox.currentSongFullPath)

self.isSongLoaded = True

self.songMetadata.read\_song\_metadata(self.songsBox.currentSongFullPath)

self.update\_slider\_bound()

self.\_\_cycle\_update\_slider\_position()

pygame.mixer.music.play(loops=0)

Він відповідає за запуск усіх необхідних ініціалізаторів потоків(очікування кінця поточного аудіо-файлу, оновлення позиції слайдера відповідно до позиції у пісні) та оновлення станів елементів графічного інтерфейсу.

Так як для роботи зі списком файлів було обрано Listbox, для орієнтації у цьому списку створено змінну, що зберігає поточний індекс файлу у ньому.

self.currentSongIndex = -1

Як і у всіх програвачів, є 4 кнопки: пауза, продовжити, попередня пісня, наступна пісня. Перші дві динамічно займають одну й ту ж позицію у інтерфейсі, підміняючи одна одну в залежності від того, чи грає пісня або чи завантажена вона до модуля програвання. Для цього, було створено спеціальні методи, що виконують роботу підміни.

**def** \_\_change\_pause\_button\_to\_play(self):

self.gui.btn\_play.grid\_remove()

self.gui.btn\_pause.grid()

**def** \_\_change\_play\_button\_to\_pause(self):

**if** len(self.songsBox.songsFullPaths) > 0:

self.gui.btn\_play.grid\_remove()

self.gui.btn\_pause.grid()

Автоматичне програвання наступної пісні досягається завдяки потоку очікування кінця пісні. Визначення події, на яку очікує потік:

self.SONG\_END = pygame.USEREVENT

pygame.mixer.music.set\_endevent(self.SONG\_END)

Метод що перевіряє чи закінчилась пісня:

**def** \_\_cycle\_wait\_for\_song\_to\_end(self):

**for** event **in** pygame.event.get():

**if** event.type == self.SONG\_END:

self.\_\_check\_if\_song\_is\_last()

self.\_\_autoplay\_next\_song()

self.stop\_wait\_for\_song\_to\_end\_cycle()

self.isSongEndedFlag = self.gui.master.after(50, self.\_\_cycle\_wait\_for\_song\_to\_end)

Можливо, якась композиція сподобається користувачу і він забажає прослухати її декілька разів. На цей випадок передбачена функція циклічного програвання одно й того ж файлу. Для перевірки встановлення режиму повторного запуску пісні створено змінну-флаг, на основі якої відбувається аналіз того, завантажувати той самий файл, чи брати наступний.

self.isSongOnRepeat = False

Важливим додатком до можливостей програвача є слайдер, який виконує так звані функції «Rewind» та «Fast forward» - перемотка назад та вперед та відображає графічно поточну позицію у пісні. Він є об’єктом класу Scale модуля ttk, що належить до Tkinter. Як було сказано раніше, він починає свою роботу після виклику методу play\_selected\_song() класу PlaybackControls, який запускає його потік.

**def** \_\_cycle\_update\_slider\_position(self):

**if** **not** self.isSongPaused:

self.songMetadata.song\_playtime\_counter()

self.gui.slider\_song.config(value=self.songMetadata.songRawPlaytime)

self.sliderPlaybackUpdateFlag = self.gui.master.after(250, self.\_\_cycle\_update\_slider\_position)

Метод представлений вище викликається кожні 250 мілісекунд.

Можливість перемотки досягається шляхом встановлення слухача події для об’єкта слайдера та прив’язки метода, що викликається при взаємодії:

**def** \_\_bind\_playback\_slider\_event(self):

self.gui.slider\_song.bind("<ButtonRelease-1>", **lambda** event: self.\_\_set\_song\_from\_new\_position(e=event,new\_position=self.gui.slider\_song.get()))

**def** change\_song\_position\_on\_slide(self, x):

self.stop\_slider\_update\_cycle()

self.songMetadata.wasSliderUsed = True self.songMetadata.update\_playtime\_on\_slide(self.gui.slider\_song.get())

Гучність вихідного сигналу є важливою складовою задоволення від прослуховування аудіо. Для її регулювання передбачено вертикальний слайдер, На відміну від попереднього слайдера, цей не потребує окремого потоку перевірки стану зовнішніх факторів. Для роботи з ним також встановлено слухач події та метод:

**def** \_\_bind\_sound\_slider\_event(self):

self.gui.slider\_volume.bind("<ButtonRelease-1>", **lambda** event: self.\_\_set\_previous\_volume(e=event,previous\_volume=self.gui.slider\_volume.get()))

**def** \_\_set\_volume(self, x):

new\_volume = self.gui.slider\_volume.get()

new\_volume\_int = int(new\_volume)

self.gui.volumeLabel.config(text=str(new\_volume\_int))

self.\_\_change\_volume\_icon\_on\_value(new\_volume\_int)

new\_volume\_converted = new\_volume / 100

pygame.mixer.music.set\_volume(new\_volume\_converted)

## Опис механізму роботи з метаданими

Для того, аби розуміти, як працювати з тією чи іншою піснею, програвачу необхідно знати про неї низку речей. Файли з розширенням «.mp3» використовують систему ID3 тегів, що являють собою метадані, розміщені на початку або в кінці файлу. Остання версія даної системи має назву ID3v2. Серед списку полів, нас цікавлять всього декілька – довжина пісні у мілісекундах, ім’я автора, назва композиції та зображення обкладинки.

Робота з цими даними проводиться за допомогою методів створеного класу SongMetadata. Довжина пісні зчитується методами модуля mutagen.mp3, та для відображення форматується модулем time.

**def** get\_song\_length(self):

song\_mutagen = mtg.MP3(self.songsBox.currentSongFullPath)

self.songRawLength = song\_mutagen.info.length

**if** self.songRawLength > 3600:

self.songFormattedLength = time.strftime("%H:%M:%S", time.gmtime(self.songRawLength))

**return**

self.songFormattedLength = time.strftime("%M:%S", time.gmtime(self.songRawLength))

Оскільки під час програвання пісні, дані про позицію представлені у мілісекундах, створено окремий метод для конвертації у нормальний формат.

**def** raw\_playtime\_to\_formatted(self, playtime):

**if** self.songRawLength > 3600:

self.songFormattedPlaytime = time.strftime("%H:%M:%S", time.gmtime(playtime))

**else**:

self.songFormattedPlaytime = time.strftime("%M:%S", time.gmtime(playtime))

Усі інші дані дістаються за допомогою модуля stagger.

**def** read\_song\_metadata(self, song\_full\_path):

self.cancel\_all\_cycles()

**try**:

self.songTag = stg.read\_tag(song\_full\_path)

**except** stg.errors.NoTagError:

self.songTitle = song\_full\_path[song\_full\_path.rfind("/") + 1:-4]

self.songFullTitle = self.\_\_build\_song\_full\_title("", self.songTitle)

**else**:

self.songTitle = self.songTag.title

self.songFullTitle = self.\_\_build\_song\_full\_title(self.songTag.artist, self.songTitle)

self.gui.songNameLabel.configure(text=self.songFullTitle)

**if** len(self.songFullTitle) > 27:

self.\_\_cycle\_spin\_song\_full\_title()

self.\_\_get\_song\_cover\_image()

self.get\_song\_length()

Метод для отримання зображення обкладинки:

**def** \_\_get\_song\_cover\_image(self):

**if** **not** self.\_\_check\_if\_tag\_has\_image():

self.gui.coverLabel.configure(image=self.gui.coverImagePlaceholder)

**return**

byte\_data = self.songTag[stg.id3.APIC][0].data

image\_io = io.BytesIO(byte\_data)

image\_file = Image.open(image\_io)

image\_file = image\_file.resize((150, 150), Image.ANTIALIAS)

self.coverRawImage = ImageTk.PhotoImage(image\_file)

self.gui.coverLabel.configure(image=self.coverRawImage)

Підготовки до запуску наступного файлу, потребує очистки попередніх даних. Для цих цілей створено окремий метод:

**def** clear\_song\_metadata(self):

self.gui.coverLabel.configure(image=self.gui.coverImagePlaceholder)

self.gui.songNameLabel.configure(text="N/A")

self.gui.songTimeLabel.configure(text="X:XX/X:XX")

self.songFullTitle = ""

self.songTag = None

self.coverRawImage = None

self.songFormattedPlaytime = None

self.songFormattedLength = None

self.songRawLength = 0

self.set\_playtime\_to\_zero()

self.stop\_song\_title\_cycle()

# Реалізація інтерфейсу

На рівні із реалізацією програмної логіки знаходиться реалізація інтерфейсу. Він є «обличчям» застосунку – тим, із чим безпосередньо контактує користувач, тому йому потрібно приділяти не менше уваги.

## Структура головного вікна

За програмування інтерфейсу відповідає клас GUI. Він містить об’єкти, що відповідають за віджети, їх налаштування та розміщення.

Усі використовувані кольори представлені статичним полями класу:

backgroundColor = "#2B2B2B"

songsListBackgroundColor = "#3C3F41"

foregroundColor = "#BBBBBB"

foregroundActiveColor = "#F0F0F0"

Як було показано у розділі «[Створення дизайну головного вінка програми](#_Створення_дизайну_головного)», основа головного вікна складається з трьох фреймів:

self.topFrame = tk.Frame(master, background=GUI.backgroundColor)

self.topFrame.grid(row=0, column=0, sticky="nswe")

self.songsListFrame = tk.Frame(master, background=GUI.backgroundColor)

self.songsListFrame.grid(row=1, column=0, sticky="nswe")

self.footerFrame = tk.Frame(master, background=GUI.backgroundColor)

self.footerFrame.grid(row=2, column=0, sticky="nswe")

Позиціонування їх самих та дочірніх контейнерів представлено на рисунку 10. Рівні вкладеності позначаються наступними кольорами:

1. червоний – рівень 0;
2. зелений – рівень 1;
3. жовтий – рівень 2.

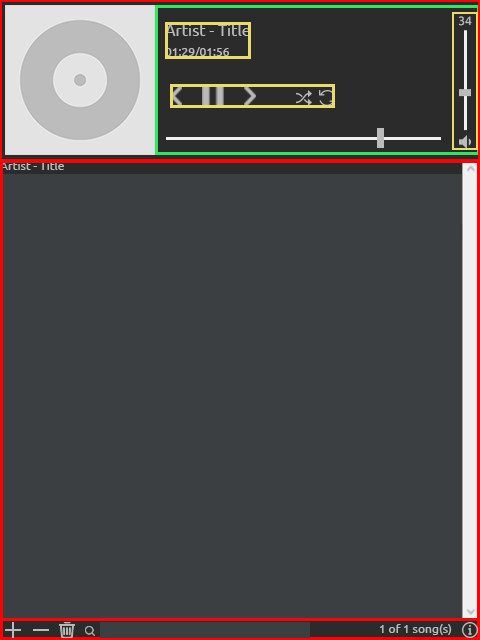


Рисунок 10 – Позиціонування фреймів

Їх розміщення виконується за допомогою системи grid. Дочірні елементи використовують як grid, так і pack.

Наприклад, так виглядає код створення та розміщення кнопки додавання файлів:

Створюються об'єкти PhotoImage, що містять зображення для різних станів віджета:

self.img\_add\_songs = tk.PhotoImage(file="../img/icons/plus.png")

self.img\_add\_songs\_hover = tk.PhotoImage(file="../img/icons/plus(hover).png")

Сам віджет:

self.btn\_add\_songs = tk.Button(self.footerFrame,

image=self.img\_add\_songs,

bg=GUI.backgroundColor,

activebackground=GUI.backgroundColor,

relief=tk.GROOVE,

bd=0)

Додавання подій, що відповідають за зміну стану:

self.btn\_add\_songs.bind("<Enter>",

**lambda** event, image=self.img\_add\_songs\_hover: self.on\_enter(e=event, image=image))

self.btn\_add\_songs.bind("<Leave>",

**lambda** event, image=self.img\_add\_songs: self.on\_leave(e=event, image=image))

Розміщення:

self.btn\_add\_songs.pack(side=tk.LEFT, ipadx="5px", ipady="2px")

Подібний цикл виконується для всіх елементів інтерфейсу.

Кнопки містять у собі зображення формату «.png», бібліотека яких зображена на наступному рисунку.

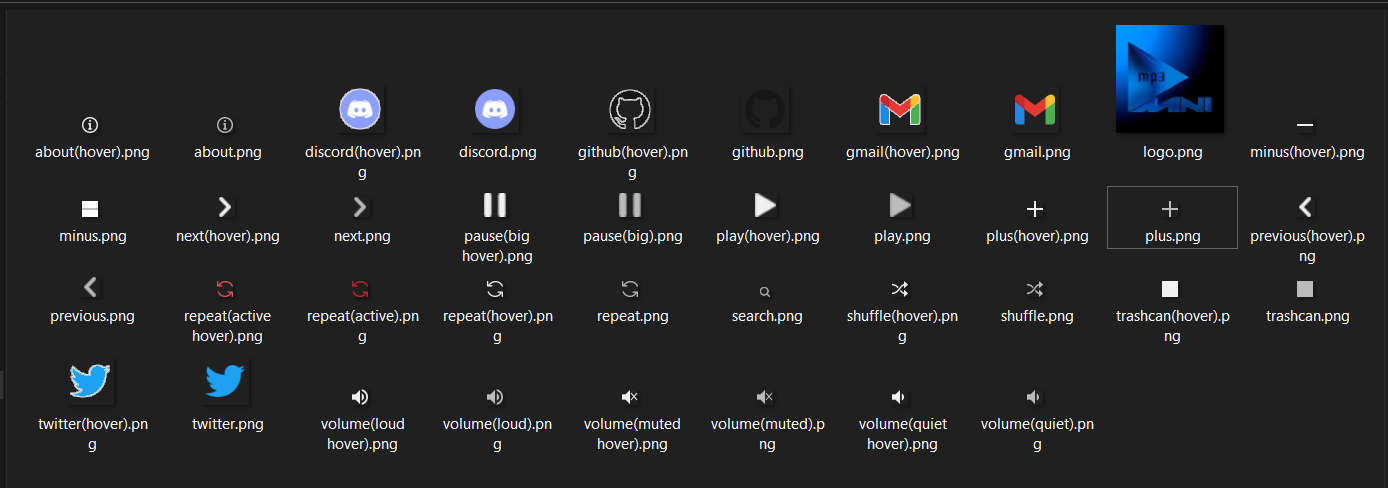


Рисунок 11 – Бібліотека зображень

# тестування готового застосунку

## Маніпуляції зі списком файлів

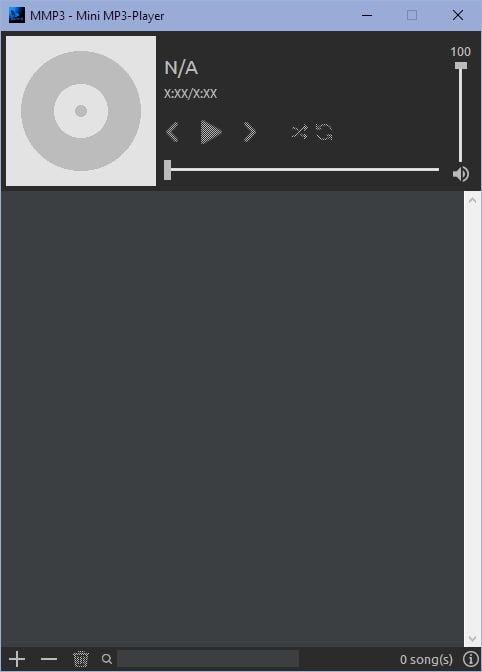


Рисунок 12 – До завантаження файлів

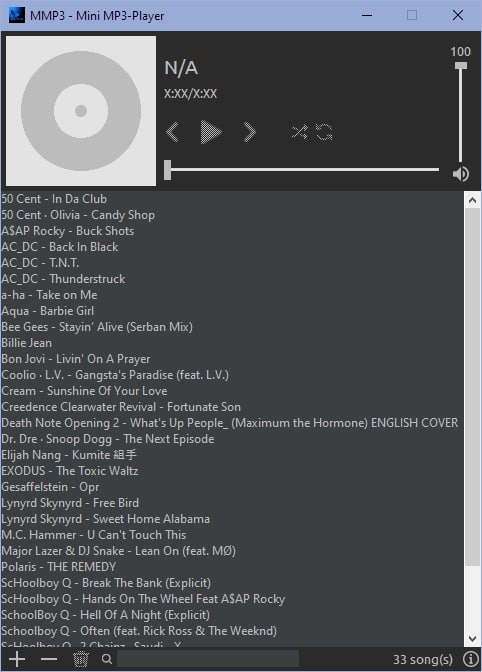


Рисунок 13 – Після завантаження

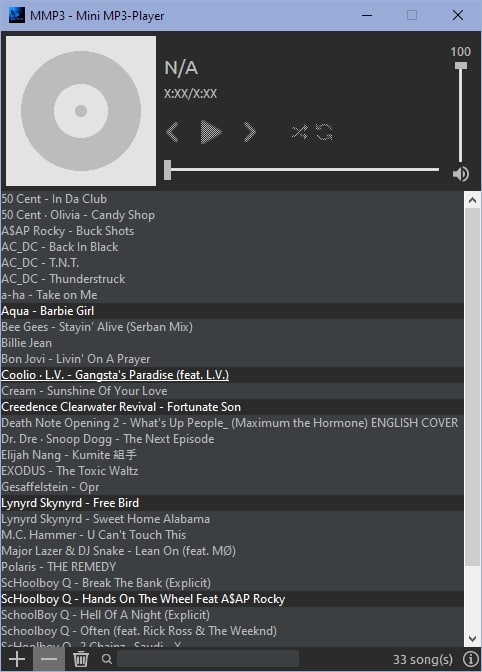


Рисунок 14 – Виділення файлів для видалення



Рисунок 15 – Після видалення

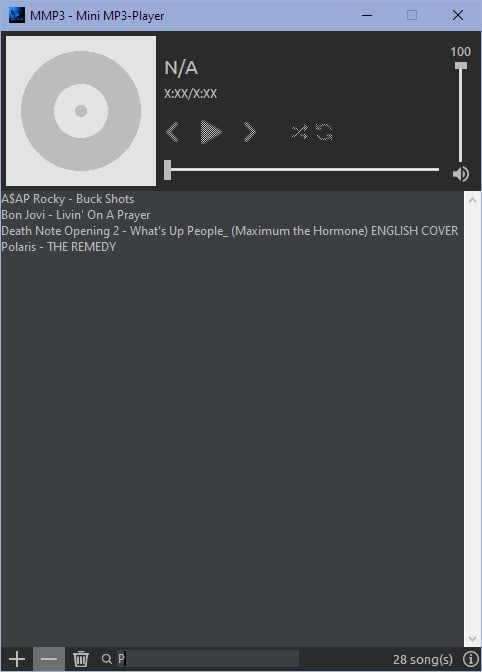


Рисунок 16 – Пошук у списку

## Програвання аудіо

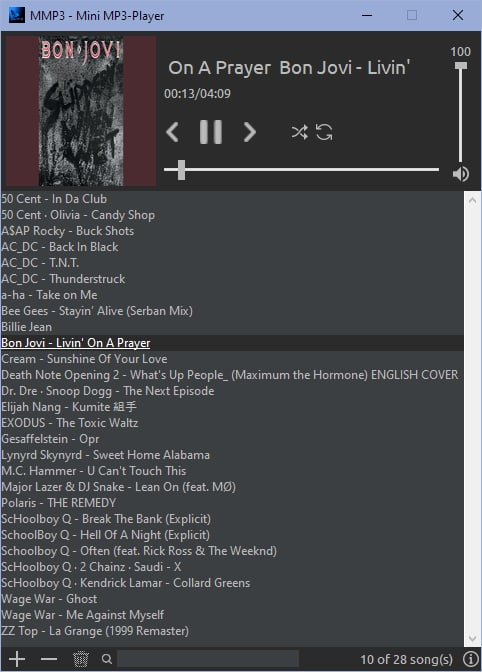


Рисунок 17 – Вибір файлу для програвання

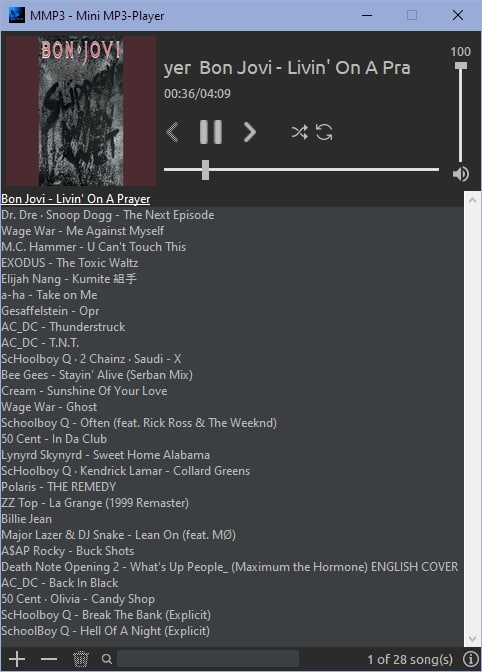


Рисунок 18 – Перемішування файлів

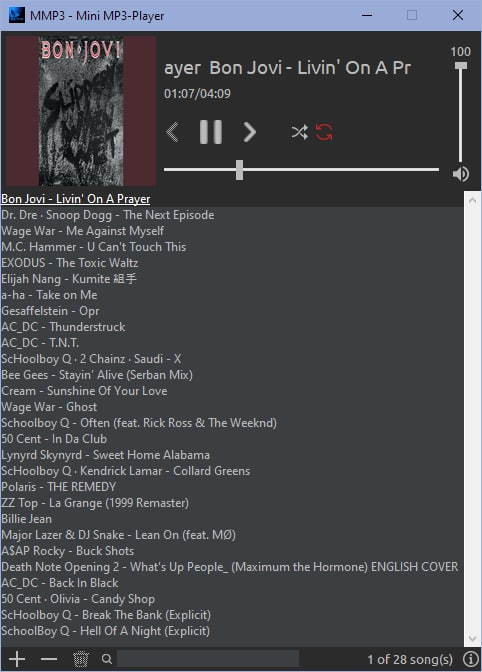


Рисунок 19 – Режим повторного програвання

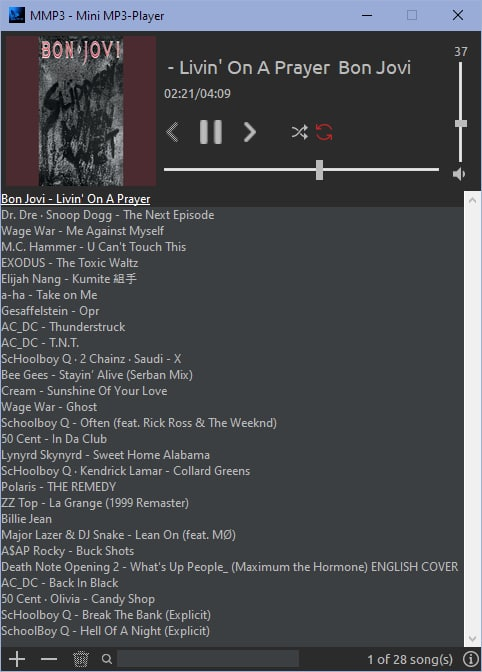


Рисунок 20 – Зміна рівня гучності

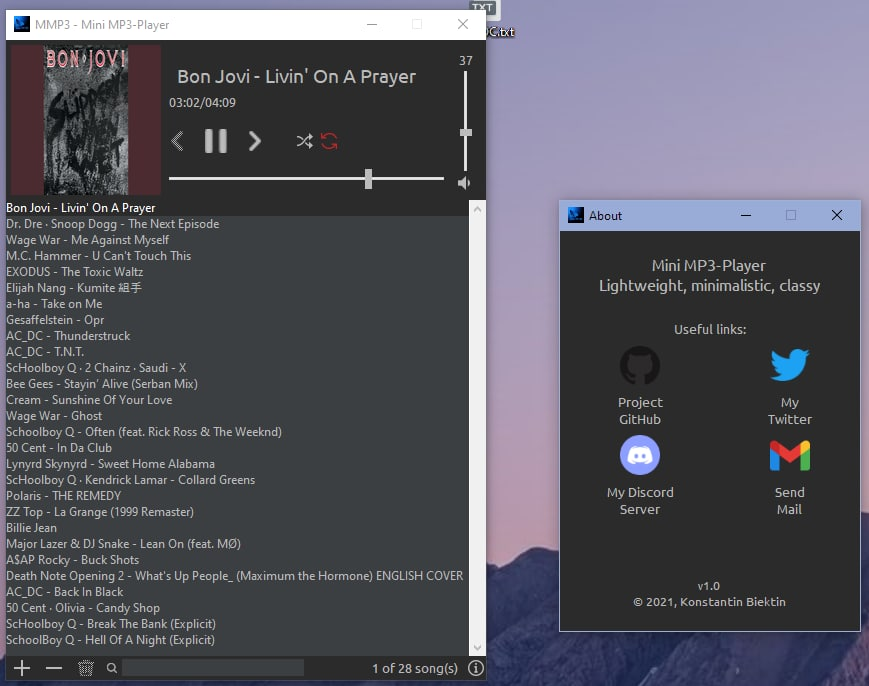


Рисунок 21 – Вікно «About»

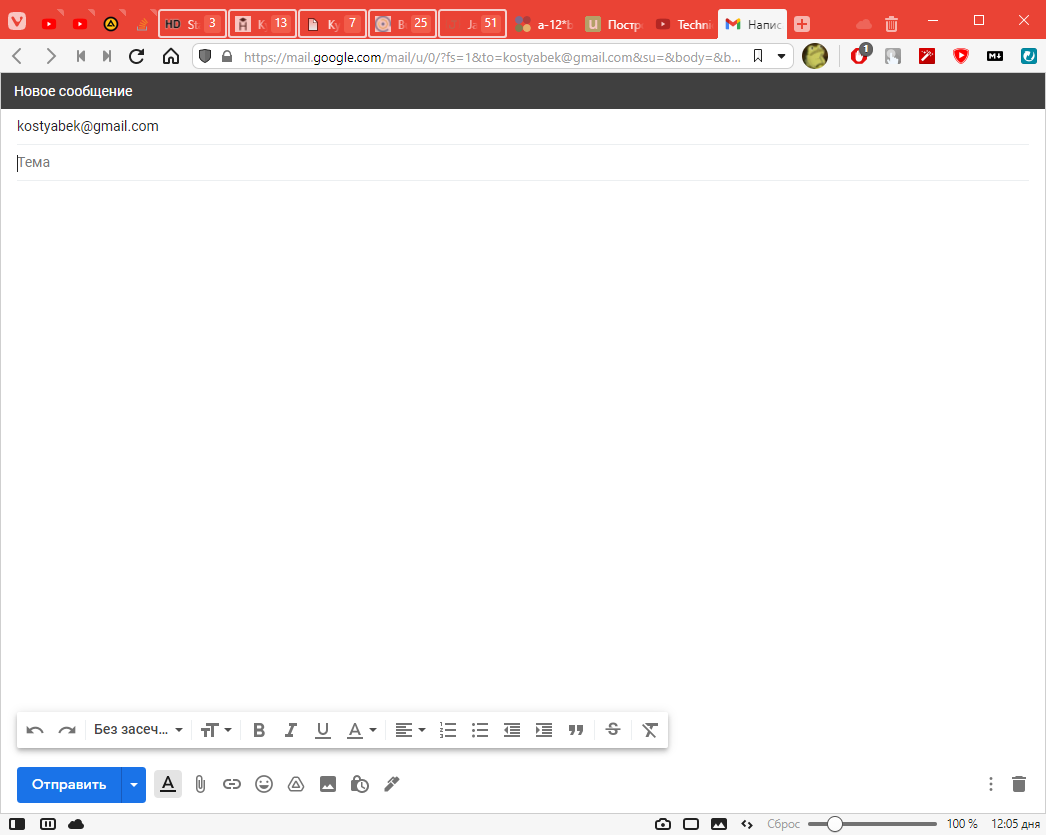


Рисунок 22 – Результат натиснення на кнопку для пошти

# висновки

Отже, в результаті виконання курсової роботи було досягнуто поставленої цілі, а саме – створено мінімалістичний MP3-програвач на мові Python. Пройдено усі етапи циклу розробки програмного забезпечення, починаючи з проєктування дизайну та функціоналу застосунку і закінчуючи програмною реалізацією та тестуванням готового продукту.

Під час розробки було застосовано навички отримані у результаті проходження курсу з мови Python у минулому семестрі. Також, проведено велику кількість самостійних досліджень у мережі Інтернет для отримання додаткових знань та прийомів у розробці на даній мові програмування.

# список використаних джерел

1. Інтернет-ресурс: <https://www.youtube.com/channel/UCFB0dxMudkws1q8w5NJEAmw> (дата звернення: 14.01.2021)

2. Інтернет-ресурс: <https://www.pygame.org/docs/ref/mixer.html> (дата звернення: 03.02.2021)

3. Інтернет-ресурс: <https://docs.python.org/3/library/tkinter.ttk.html#ttk-styling> (дата звернення: 24.03.2021)

4. Інтернет-ресурс: <https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html> (дата звернення: 07.04.2021)

5. Інтернет-ресурс: <https://www.tutorialspoint.com/python/index.htm> (дата звернення: 11.02.2021)

6. Марк Лутц. Learning Python. Fifth Edition. Пекін, Бостон, Фарнхем, Севастополь, Токіо, 2019. 269-303 с.

7. Лучано Рамальо. Fluent Python. Пекін, Кембрідж, Фарнхем, Кельн, Севастополь, Токіо, 2016. 248-276 с.

8. Зед Шоу. Learn Python the Hard Way. 3rd edition. Бостон, 2014. 194-199 с.

9. Джон Грейсон. Python and Tkinter Programming. Шелтер Айленд, 2000. 338-348 с.

10. Біл Любанович. Introducing Python: Modern Computing in Simple Packages. Пекін, Бостон, Фарнхем, Севастополь, Токіо, 2015. 71-111 с.

**ЗВІТ**

про унікальність курсової роботи на тему:

« MP3-програвач на мові Python »

студента спеціальності 121 Інженерія програмного забезпечення, 209 групи

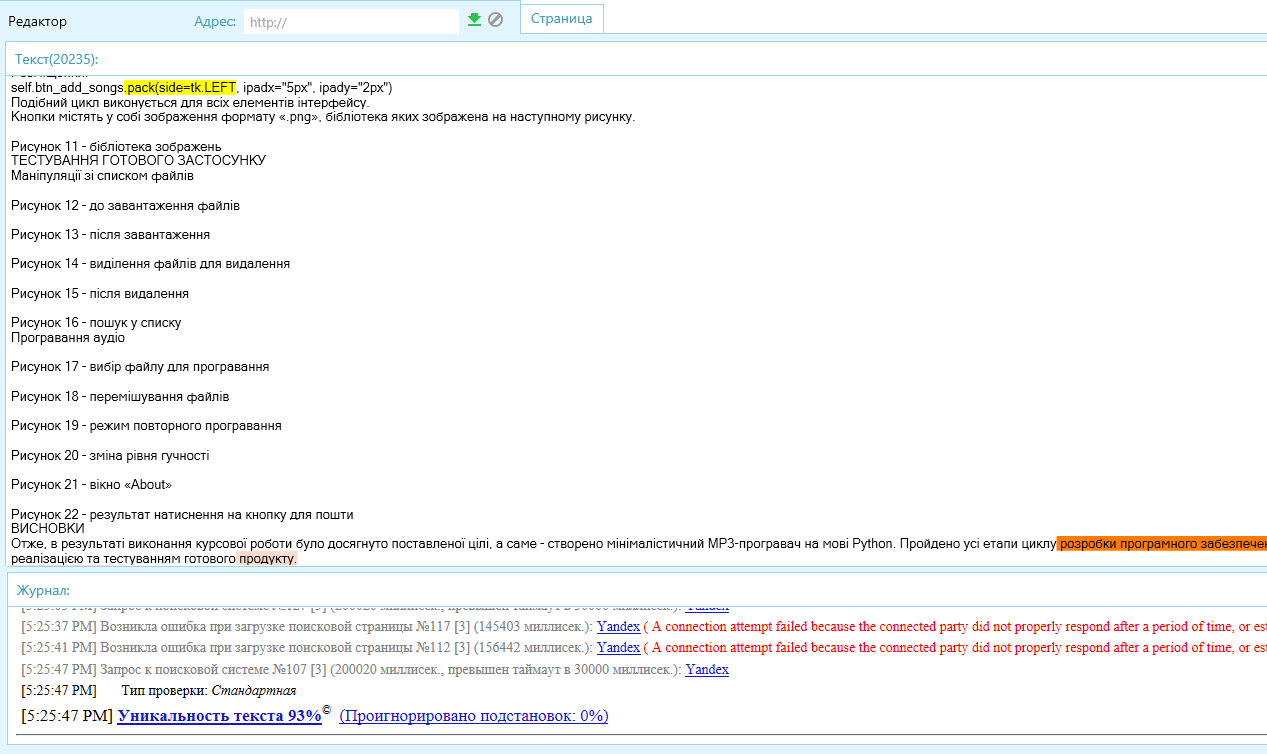
|  |
| --- |
| Бєктін К.О. |
| прізвище, ім’я, по-батькові |

|  |  |
| --- | --- |
| Шифр роботи: | 121 – КР.ПЗ.00 – 209.1910901 |

|  |  |
| --- | --- |
| Перевірку тексту здійснено сервісом: | Програма «AntiPlagiarism.NET» |
|  |  |

Результат перевірки тексту курсового проекту на унікальність складає 93%.

**Скріншот** результату  наведений на рисунку нижче.



Ст. викладач кафедри ІПЗ                                                  С.Ю. Боровльова

|  |  |  |  |
| --- | --- | --- | --- |
| Дата: | « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 2021 р. |  |

Посилання на github-репозиторій проєкту: <https://github.com/kostyabek/mmp3>