Studierea bibliotecii PyQt5 pentru limbajul de programare Python si crearea unui aplicatii de tip music player

ELABORAT: ST.GR.TI-194, ZAVOROT DANIEL

Analiza domeniului

- Aplicația care a fost realizata este de tip desktop creata in limbajul Python
- Pentru realizarea acestei aplicații s-a utilizat biblioteca Qt (PyQt5). De asemenea a fost folosita aplicatia Qt Designer ce permite crearea GUI-urilor in real-time cu ajutorul functiei (drag ‘n drop).

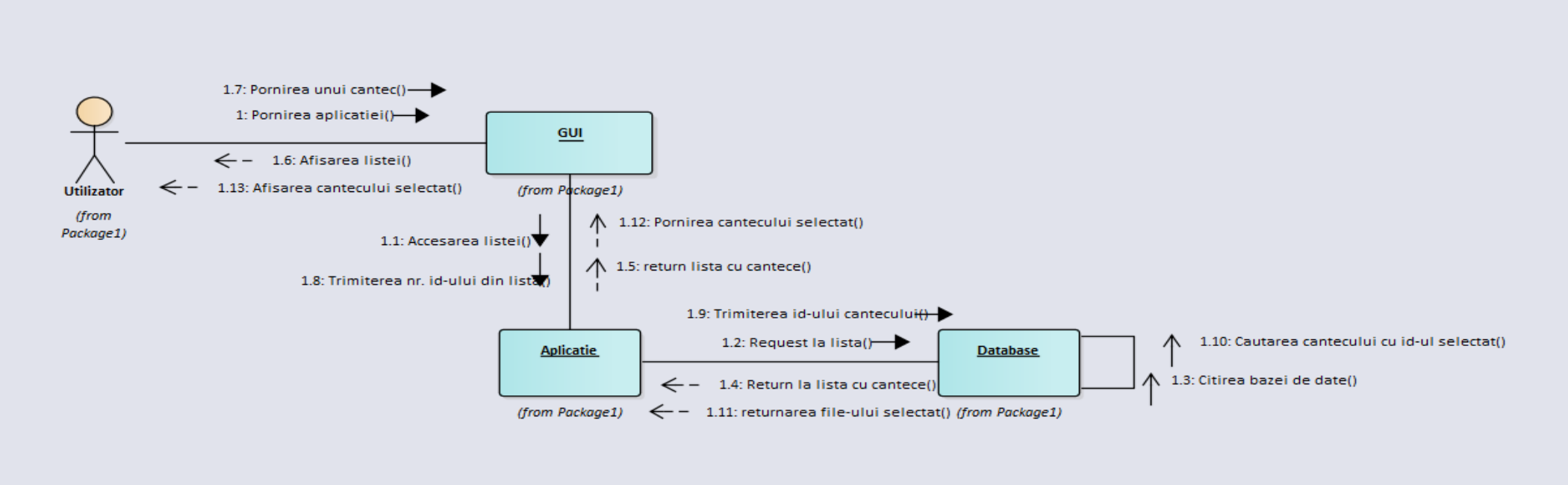
Scopul si Obiectivele

Principalul scop al acestui proiect este crearea unei aplicații de tip music player cu un funcțional de baza mediu. Design-ul va fi de culoare inchisa, cu culori predominante negru/gri/verde. De asemenea va avea window bar-ul sau propriu, care va fi de culoare inchisa, deoarece windows nu permite schimbarea bar-ului care sta la baza tuturor aplicatiilor. Dupa functional va avea incarcarea, stergerea, editarea fisierilor cu extensia .mp3. De asemenea va avea posibilitatea incarcarei unui cover pentru piesa aleasa. Ca si orice alt player pentru muzica, va avea posibilitatea de controlare a pieselor, adica va contine shuffle mode, repeat once, repeat this si desigur normal mode. Va contine slide pentru volum si un slide pentru controlarea cantecului curent.

Obiectivele de baza sunt:

1. Realizarea unei aplicații de tip music player
2. Optimizarea maximala a codului
3. Executarea non-stop a programului chiar daca a aparut vreo greseala

Imaginea generala a sistemului



Interacțiunea user-ului cu sistemul cu ajutorul GUI-ului realizat

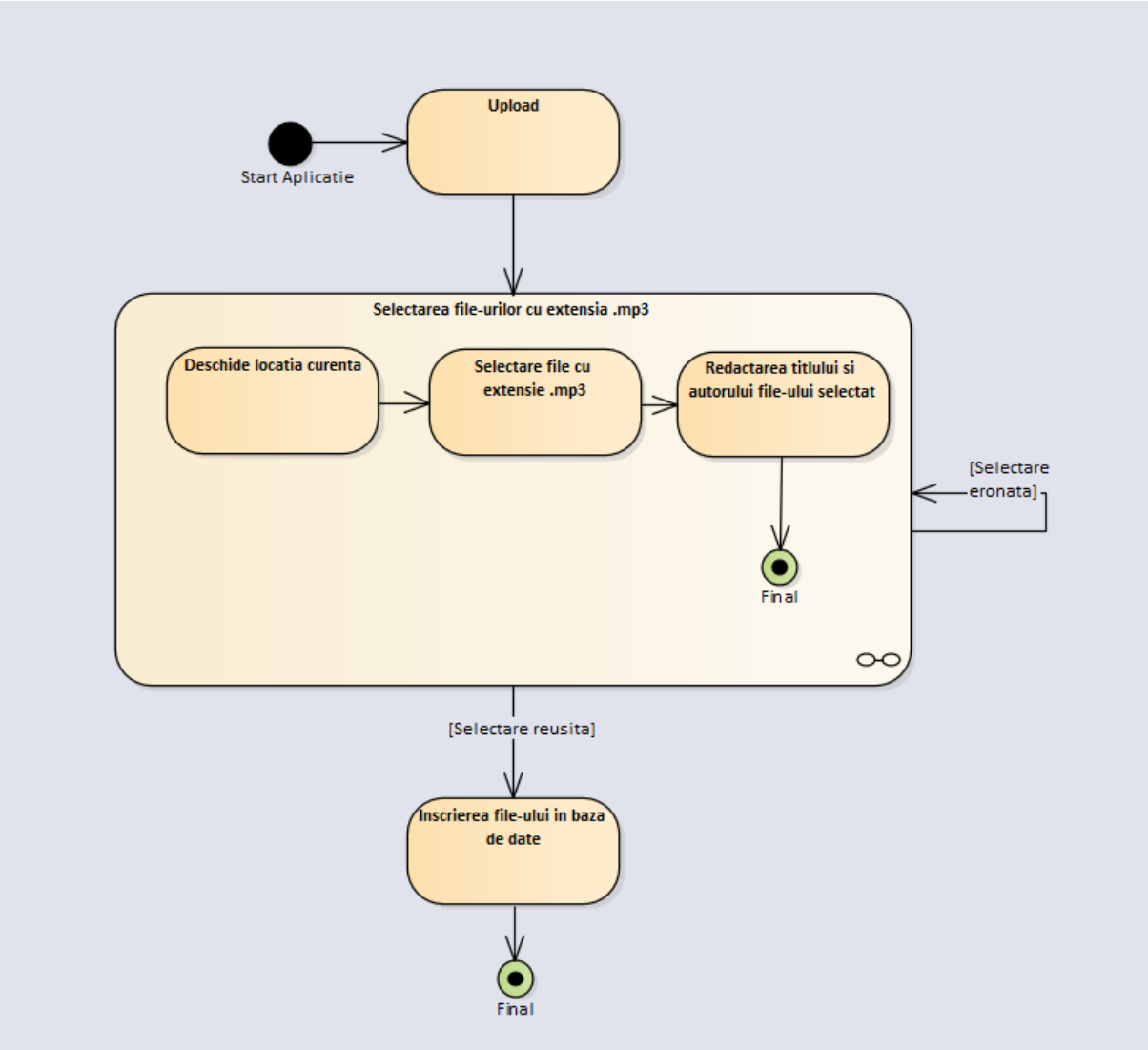


Diagrama de stare a procesului de selectare file

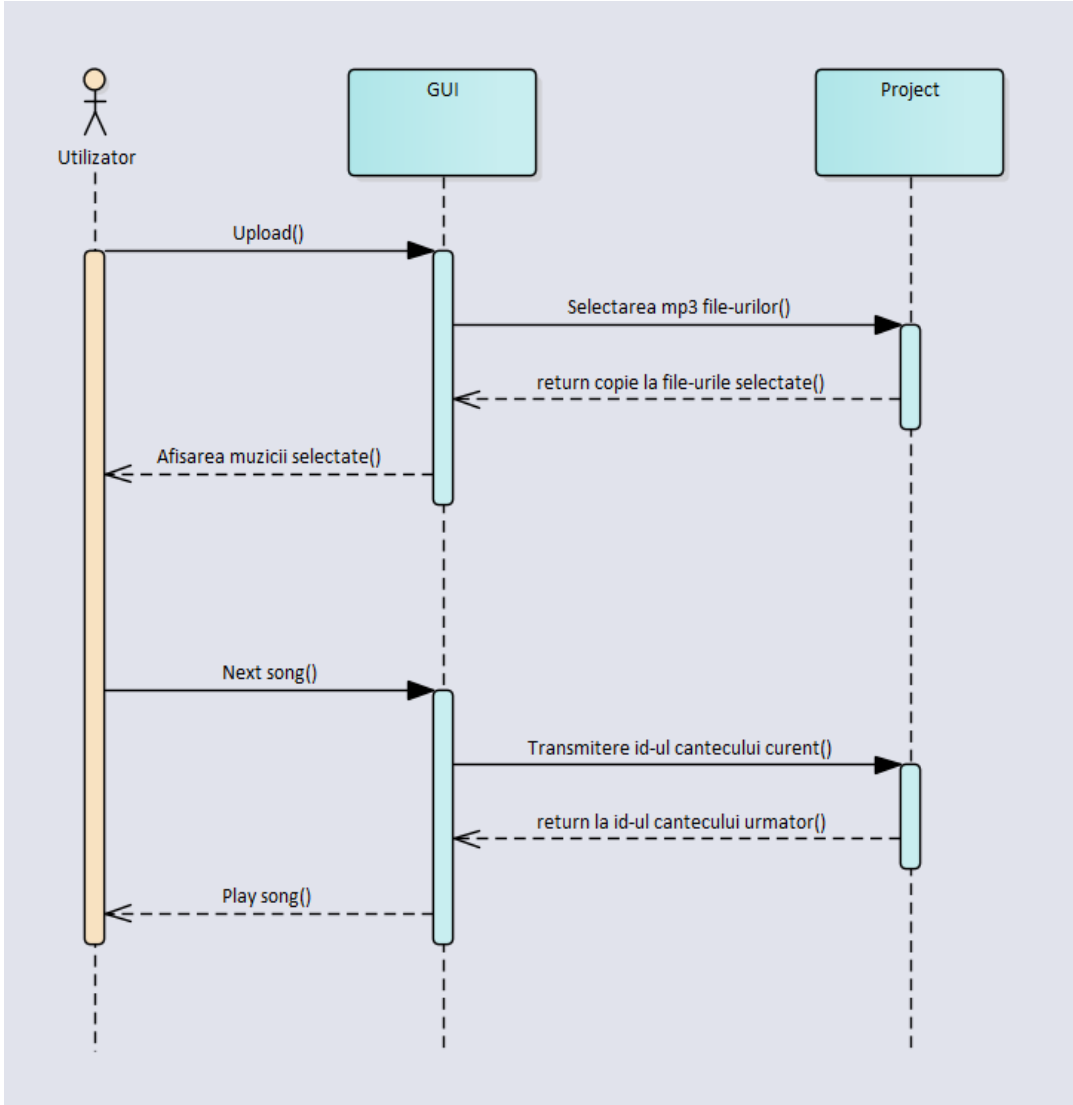
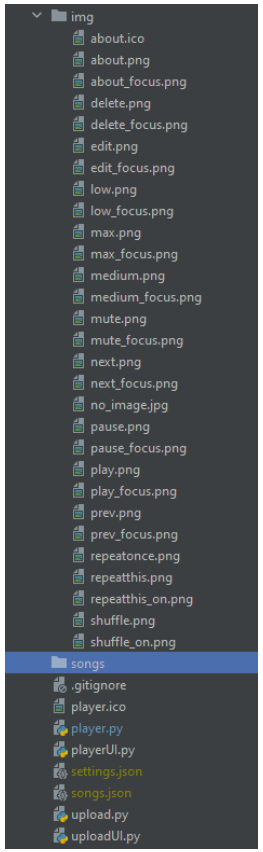


Diagrama de secvență a incarcarii unui cantec in aplicatie

Structura proiectului



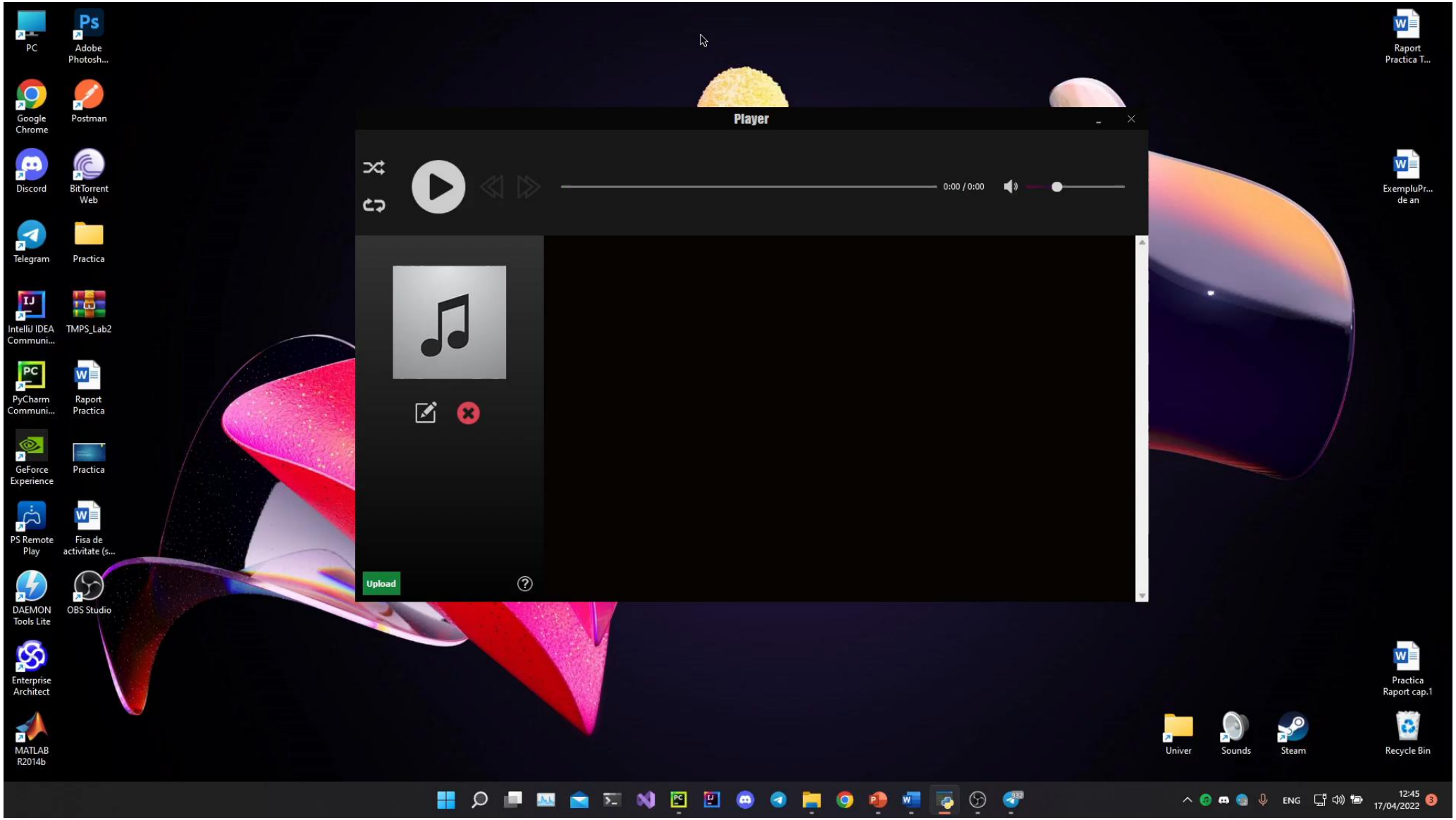

```python
# read songs from songs.json
def read_songs_from_json(self):
    if not os.path.exists('songs'):
        os.makedirs('songs')
    self.player = QMediaPlayer()
    self.playlist = QMediaPlaylist(self.player)
    try:
        with open("songs.json", "r", encoding="utf-8") as file:
            data = json.load(file)
        self.titles.clear()
        self.artists.clear()
        self.covers.clear()
        for i in data["Songs"]:
            # title
            self.titles.append(i["title"])
            # artist
            self.artists.append(i["artist"])
            # cover
            if i["cover"] == "Undefined":
                self.covers.append("no_image.jpg")
            else:
                self.covers.append(i["cover"])
    except Exception as e:
        print(e)
    self.read_files_songs()
```

```json
{
    "Songs": [
        {
            "id": 0,
            "title": "All Night",
            "artist": "2Scratch",
            "cover": "Undefined"
        },
        {
            "id": 1,
            "title": "Sober",
            "artist": "2Scratch",
            "cover": "Undefined"
        },
        {
            "id": 2,
            "title": "Channel Dark Side",
            "artist": "Blind",
            "cover": "Undefined"
        },
        {
            "id": 3,
            "title": "Livin' la Vida Loca",
            "artist": "COMETRON",
            "cover": "Undefined"
        },
```

Instrumente si tehnologii utilizate

Pentru realizarea acestei aplicatii am ales limbajul de programare Python si biblioteca PyQt5. De asemenea am folosit file-urile cu extensia .json care servesc ca baza de date NoSQL pentru aplicatia data. Nu in ultimul rand a fost utilizat aplicatia Qt Designer care permite crearea GUI-ului mai usor, de asemenea GUI creat in C++ putem sa-l convertim pentru limbajul Python cu ajutorul unei comenzi in consola, fiind un foarte mare plus pentru programatorii ce utilizeaza aceasta librarie.

Video prezentare



Multumesc pentru atentie!

MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport

Disciplina: Practica Tehnologică

Tema: Studierea bibliotecii PyQt5 pentru limbajul de programare Python si crearea unui aplicatii de tip music player
(Studying the PyQt5 library for the Python programming language and creating a music player application)

Student: _____ **Zavorot Daniel, TI-194**

Conducator: _____ **Andrievschi-Bagrin Veronica, asis. univ.**

Chişinău, 2022

Cuprins

Introducere

Ce este un music player? Un music player este o aplicatie hardware sau software care redă fișiere audio codificate în MP3 și alte formate audio. Pe partea de software, aplicațiile care se află în computerul utilizatorului, cum ar fi iTunes, Windows Media Player și RealPlayer, sunt folosite pentru a organiza o colecție de muzică, a reda fișiere audio și a extrage muzică de pe un CD. Playerele software pot oferi, de asemenea, acces la posturi de radio pe Internet și la alte site-uri audio de streaming.

Pe partea hardware, dispozitivele folosesc memorie flash cu stare solidă pentru a stoca melodiile descărcate dintr-un magazin online sau de pe computerul utilizatorului. iPod-ul Apple a fost de multă vreme liderul industriei, dar playerele au venit de toate dimensiunile, cu unele unități inclusiv radio FM. Vezi iPod.

Toate playerele, fie hardware sau software, acceptă MP3 și, în general, mai multe alte formate. AAC este formatul recomandat de Apple, iar FLAC oferă sunet de calitate CD. Vedeți exemple de codec, MP3, AAC, FLAC, Pono și audio de înaltă rezoluție.

O aplicație de redare muzicală este încorporată în toate smartphone-urile și tabletele, precum și în multe receptoare radio prin satelit și sisteme de navigație în bord.

Lucrarea dată urmărește scopul de a elabora crearea unei aplicații simple de tip music player cu un funcțional de baza ce ar permite redarea unui fișier cu extensia .mp3.

Lucrarea este structurată în trei capitole, unde se va analiza concret modul de proiectare și de realizare a proiectului dat, tehnologiile utilizate dar si modul de realizare a unei astfel de aplicații.

Primul dintre aceste capitole vine să facă o introduce generala asupra proiectului, el cuprinde scopul, obiectivele și cerințele sistemului dat. Totodată in acest capitol se face analiza domeniului de studiu si a sistemelor deja existe de acest tip.

Al doilea capitol cuprinde partea de realizare a acestui sistem, in acest capitol are loc realizarea la nivel de cod a aplicației. Aplicația data are sa fie proiectata iar tehnologiile utilizate sa fie descrise. Descrierea codului se va face pe module. Acest capitol este unul ce tine mai mult de lucrul practic însă acest lucru are sa conțină o documentație relevanta.

În ultimul capitol se vorbește despre aplicație la general si se face o documentație a produsului realizat. Acest capitol fiind unul de informare despre modul de funcționare a produsului. Elementele de documentație a aplicației sunt: denumirea, caracteristicile, imagini sau descrierea.

1 Analiza domeniului de studiu

Aplicația face parte din domeniul tehnologiilor informaționale. Aplicația este de tip desktop având un GUI ce permite activitățile de baza. Astfel de aplicații sunt utilizate in viața cotidiana pentru activități simple (vizionarea unui film, ascultarea unui cântec) însă fără de aceste aplicații ar apărea unele dificultăți.

Pentru realizarea acestei aplicații s-a utilizat biblioteca PyQt5 pentru limbajul de programre Python.

Un avantaj al limbajului Python este existența unei ample biblioteci standard de metode. O astfel de biblioteca este biblioteca PyQt5, care a fost dezvoltata de catre developerii biblitoecii Qt, fiind un sistem inter-platformă de dezvoltare a programelor pentru calculator, care cuprinde o bibliotecă cu elemente de control, folosit atât pentru crearea programelor cu interfață grafică cât și pentru programe fără interfață grafică, cum sunt serverele. Cele mai cunoscute utilizări ale Qt sunt KDE, browserul web Opera, Google Earth, Skype, Qtopia. Qt este produs de firma norvegiană Trolltech. Prin urmare toate functionalitatile bibliotecii Qt pentru C++, sunt si in biblioteca PyQt5 pentru Python. De asemenea cu ajutorul bibliotecii PyQt5, GUI-ul creat pentru C++ este transformat pentru limbajul Python, prin urmare Qt Designer (aplicatie care permite crearea GUI-urilor pentru C++) poate fi folosita de catre developerii care programeaza in limbajul Python.

Figura 1 – Logotipul bibliotecii PyQt5

1.1 Scopul, obiectivele și cerințele sistemului

Principalul scop al acestui proiect este crearea unei aplicații de tip music player cu un funcțional de baza mediu. Design-ul va fi de culoare inchisa, cu culori predominante negru/gri/verde. De asemenea va avea window bar-ul sau propriu, care va fi de culoare inchisa, deoarece windows nu permite schimbarea bar-ului care sta la baza tuturor aplicatiilor. Dupa functional va avea incarcarea, stergerea, editarea fisierilor cu extensia .mp3. De asemenea va avea posibilitatea incarcarei unui cover pentru piesa aleasa. Ca si orice alt player pentru muzica, va avea posibilitatea de controlare a pieselor, adica va contine shuffle mode, repeat once, repeat this si desigur normal mode. Va contine slide pentru volum si un slide pentru controlarea cantecului curent.

Obiectivele de baza sunt:

- Realizarea unei aplicații de tip music player
- Optimizarea maximala a codului
- Executarea non-stop a programului chiar daca a aparut vreo greseala

Cerințele sistemului sunt:

- Aplicație GUI moderna
- Consum de memorie minim
- Posibilitatea redării fișierelor cu extensia .mp3
- Posibilitatea controlarii cantecului curent
- Posibilitatea controlarii playlist-ului

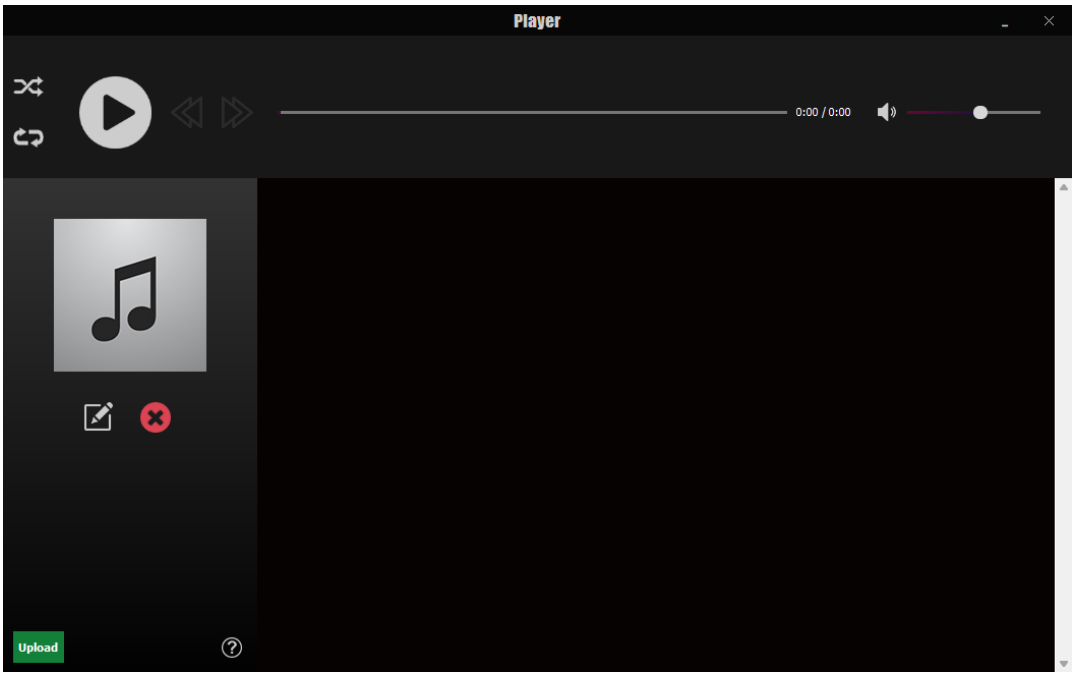


Figura 2 – GUI-ul aplicatiei

1.2 Analiza sistemelor deja existente

Sunt o multime de sisteme existente care ofera posibilitati diferite si arata diferit, insa majoritatea au integrat reclama sau o multime de functionalitati care ocupa memoria calculatorului. De asemenea sunt sisteme care sunt cu plata.

Mai jos sunt enumerate trei sisteme deja existente care fac parte din categoria proiectului meu:

- ➢ Groove Music
- ➢ Spotify
- ➢ ITunes

Primul sistem este player-ul integrat a SO Windows. Cu el m-am folosit o buna parte din timp si pot spune ca design-ul este la un nivel inalt ca si functionalitatile, insa cum am spus anterior, ocupa memorie in plus pentru toate functionalitatile lui care eu nu le folosesc. Inca un punct pozitiv ar fi ca este absolut gratis fara reclama sau ceva de genu in comparatie cu sistemele care vor urma.

Al doilea sistem este Spotify, o aplicatie cu o mare biblioteca cu muzica, fiind numarul 1 in lume in categoria sa (music player). In functionalitati are minimul necesar, si un design frumos. Ca si Groove Music, o buna parte din timp m-am folosit si de Spotify, si pot spune ca este o aplicatie limitata, adica nu poti asculta de pe local muzica ta, nu poti modifica nimic (am in vedere cantecele din biblioteca), fiecare luna trebuie sa platesti ca sa asculti muzica, daca nu atunci va aparea reclama fiecare al 2 cantec, si fiind o aplicatie care gestioneaza cantecele online, ocupa memorie fizic putin, insa memoria RAM ocupata este mai mare decat sistemul precedent.

Al treilea sistem este iTunes, o aplicatie facuta de compania Apple inc. care ca si Spotify ofera o biblioteca cu muzica larga pentru o plata lunara. Din experienta proprie pot spune ca biblioteca la iTunes este mai mica decat la Spotify, dar plata este la fel. De asemenea design-ul pentru mine este unul deja invechit. La fel ca si sistemul anterior muzica de pe local nu poate fi incarcata.

2 Proiectarea aplicației

Prin modelarea conceptuală a datelor (analiza si definirea cerintelor) se urmăreşte construirea unui model al datelor care să asigure transpunerea exactă a realităţii din domeniul analizat, fără a lua în considerare cerinţele specifice unui model de organizare a datelor (cum este modelul relaţional), criteriile de calitate privind organizarea datelor, cerinţele nefuncţionale ale sistemului şi criteriile de performanţă privind stocarea şi accesarea datelor.

UML, scurt pentru Unified Modeling Language, este un limbaj de modelare standardizat constând dintr-un set integrat de diagrame, dezvoltat pentru a ajuta dezvoltatorii de sisteme și software pentru specificarea, vizualizarea, construirea și documentarea artefactelor sistemelor software, precum și pentru modelarea afacerilor și alte sisteme non-software. UML reprezintă o colecție de cele mai bune practici de inginerie care s-au dovedit de succes în modelarea sistemelor mari și complexe. UML este o parte foarte importantă în dezvoltarea de software orientat pe obiecte și în procesul de dezvoltare a software-ului. UML folosește în mare parte notări grafice pentru a exprima proiectarea proiectelor software. Utilizarea UML ajută 15 echipele de proiect să comunice, să exploreze proiectele potențiale și să valideze designul arhitectural al software-ului.

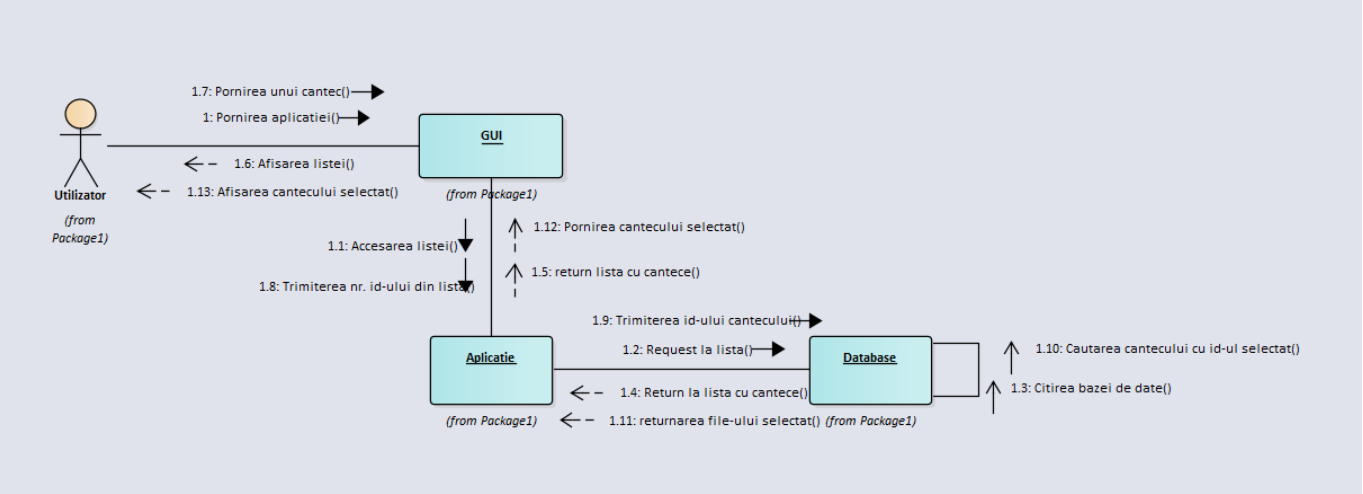


Figura 3 – Diagrama de colaborare a user-ului cu sistemul

In figura 3 este reprezentata diagrama interactiunilor utilizatorului cu sistemul dat. Utilizatorul poate interactiona cu sistemul dat cu ajutorul GUI-ului, care trimite informatia cu ajutorul id-urilor spre aplicatie care comunica cu baza de date (file songs.json). De asemenea este reprezentat modul de pornire a unei piesei cu ajutorul GUI-ului.

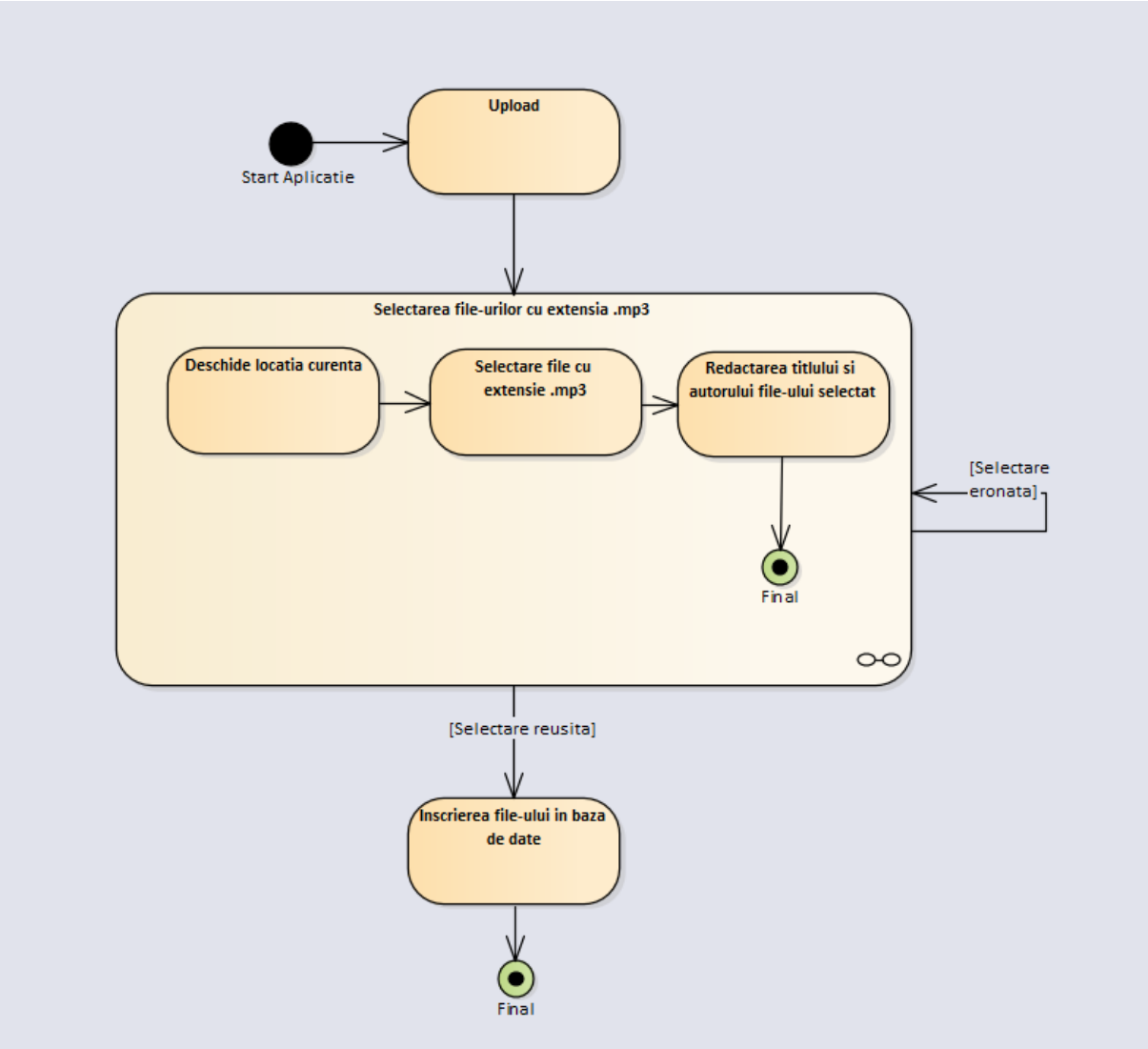


Figura 4 - Diagrama de stare a procesului de selectare file

In Figura 4 este reprezentata diagrama de stare a sistemului pentru procesul de selectarea a unui file cu extensia .mp3. După accesarea meniului are loc un state machine pentru selectarea file-ului cu extensia permisa pentru redare, in cazul in care selectarea este eronata procesul se începe de la început. Iar in caz contrar, file-ul cu extensia .mp3 este copiat in folder-ul „songs” si inscris in baza de date (songs.json).

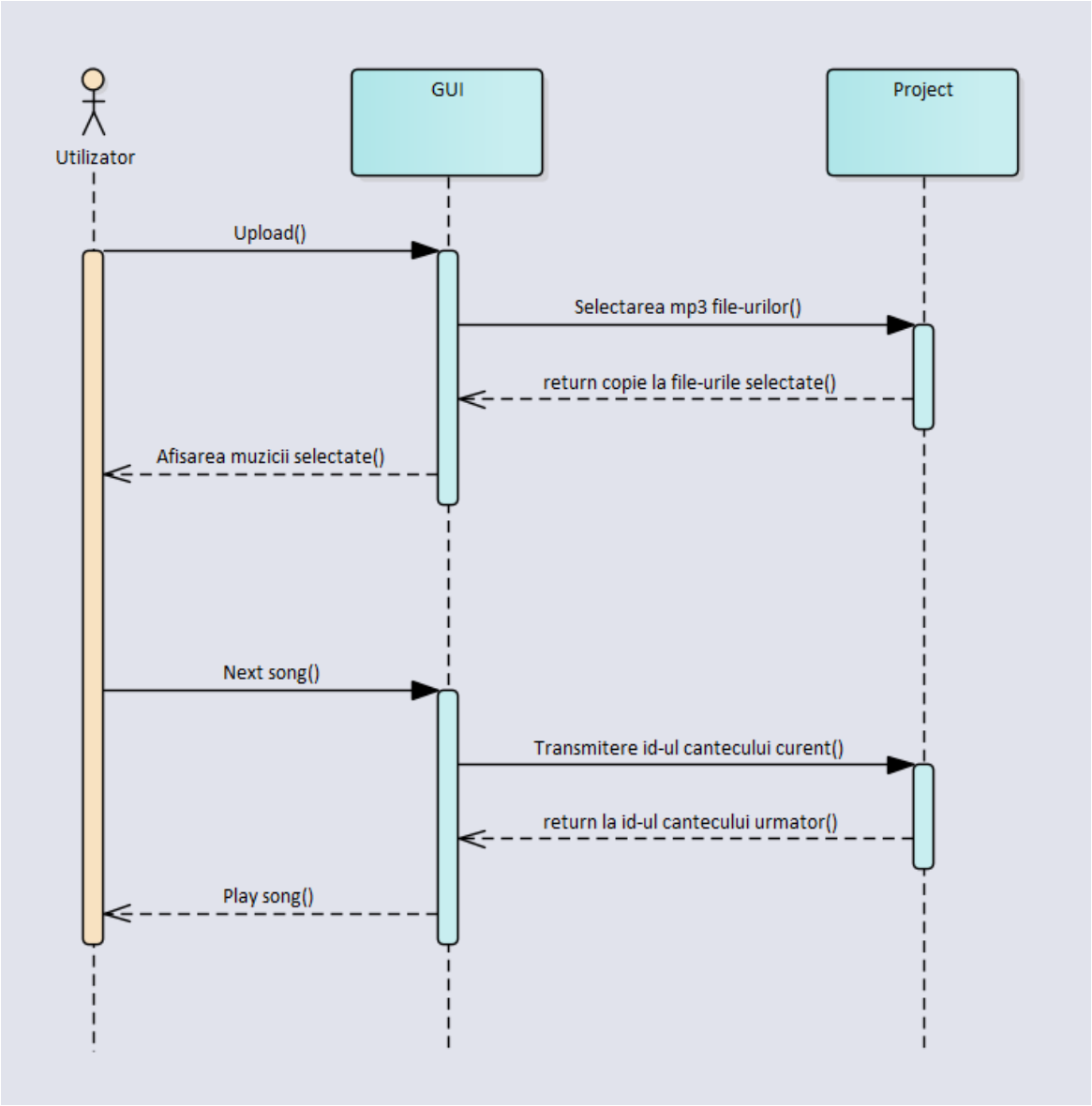


Figura 5 - Diagrama de secvență a incarcarii unui cantec in aplicatie

În figura 5 observăm diagrama de secvență pentru procesul de adăugare a unui cantec in playlist-ul aplicatiei. In sistema data putem observa ca mesajele sunt de tip *call* si *return*. Mesajul de tip *call* este utilizat pentru a invoca chemarea unei operațiuni. Iar mesajul de tip *return* returnează valoarea operației executate, in cazul dat daca adăugarea a fost cu succes.

2.1 Descrierea tehnologiilor pentru sistem

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C. Python este un limbaj multifuncțional folosit de exemplu de către companii ca Google sau Yahoo! pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python. Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv Linux, BSD și Mac OS X includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu (garbage collector). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

Cum am spus, un avantaj al limbajului este existența unei ample biblioteci standard de metode. O astfel de biblioteca este biblioteca PyQt5, care a fost dezvoltata de catre developerii biblitoecii Qt, fiind un sistem inter-platformă de dezvoltare a programelor pentru calculator, care cuprinde o bibliotecă cu elemente de control, folosit atât pentru crearea programelor cu interfață grafică cât și pentru programe fără interfață grafică, cum sunt serverele. Cele mai cunoscute utilizări ale Qt sunt KDE, browserul web Opera, Google Earth, Skype, Qtopia. Qt este produs de firma norvegiană Trolltech. Prin urmare toate functionalitatile bibliotecii Qt pentru C++, sunt si in biblioteca PyQt5 pentru Python. De asemenea cu ajutorul bibliotecii PyQt5, GUI-ul creat pentru C++ este transformat pentru limbajul Python, prin urmare Qt Designer (aplicatie care permite crearea GUI-urilor pentru C++) poate fi folosita de catre developeriicare programeaza in limbajul Python.

2.2 Descrierea la nivel de cod pe module

Structura proiectului la nivel de cod arata in modul următor:

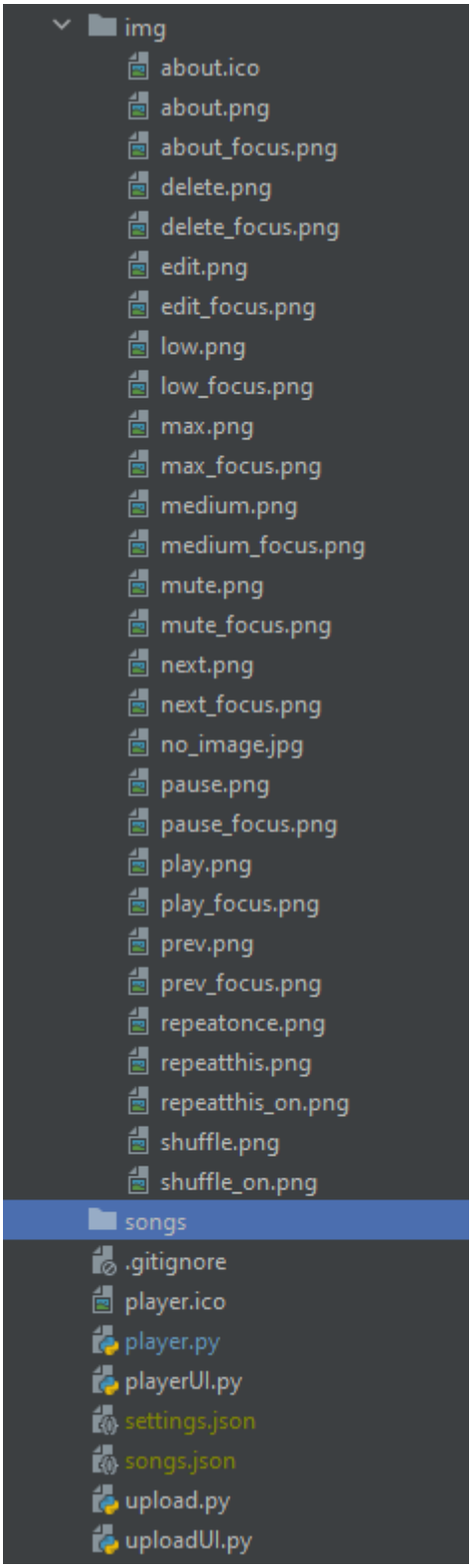


Figura 6 - Structura music player-ului

La lansarea aplicației se deschide însăși player-ul unde utilizatorul poate asculta playlist-ul care a fost incarcat, iar daca utilizatorul deschide aplicatia prima data, atunci lista cu cantece va fi goala, iar singurul buton care va functiona va fi buotnul „Upload” care se afla in stanga-jos.

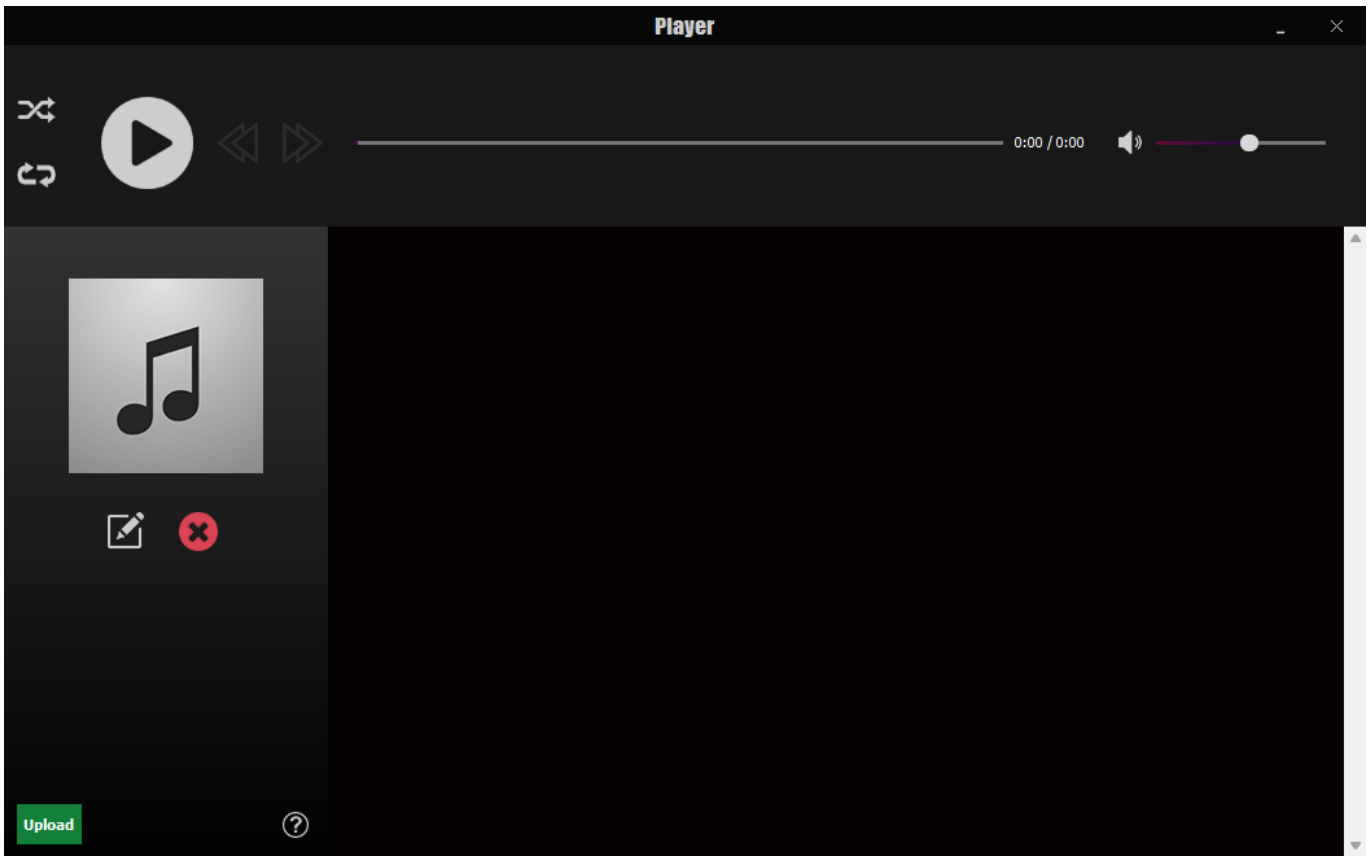


Figura 7 – Music player fara nici-un cantec

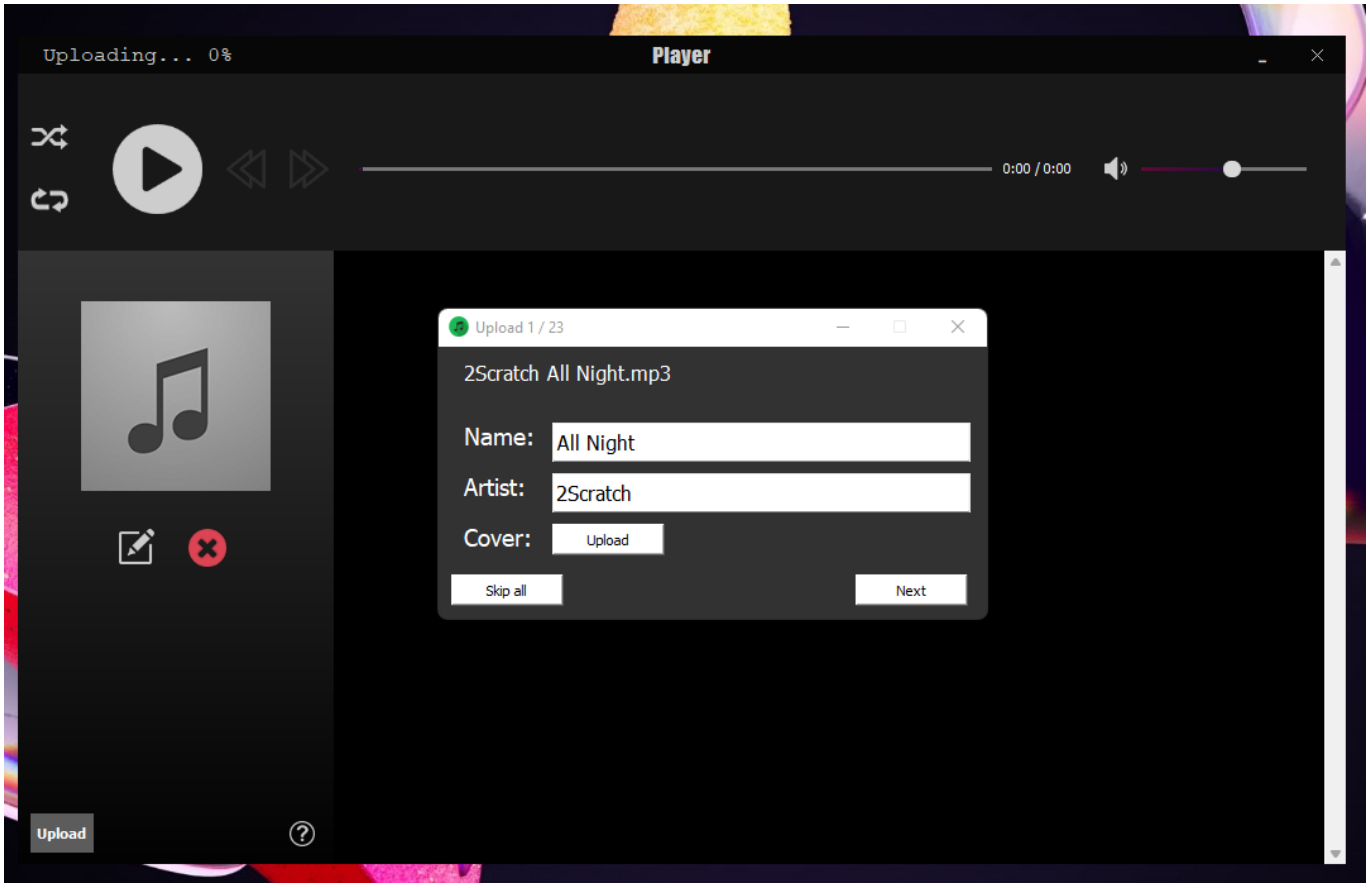


Figura 8 – Music player dupa ce am apasat butonul „Upload” si am ales file-urile cu extensia .mp3

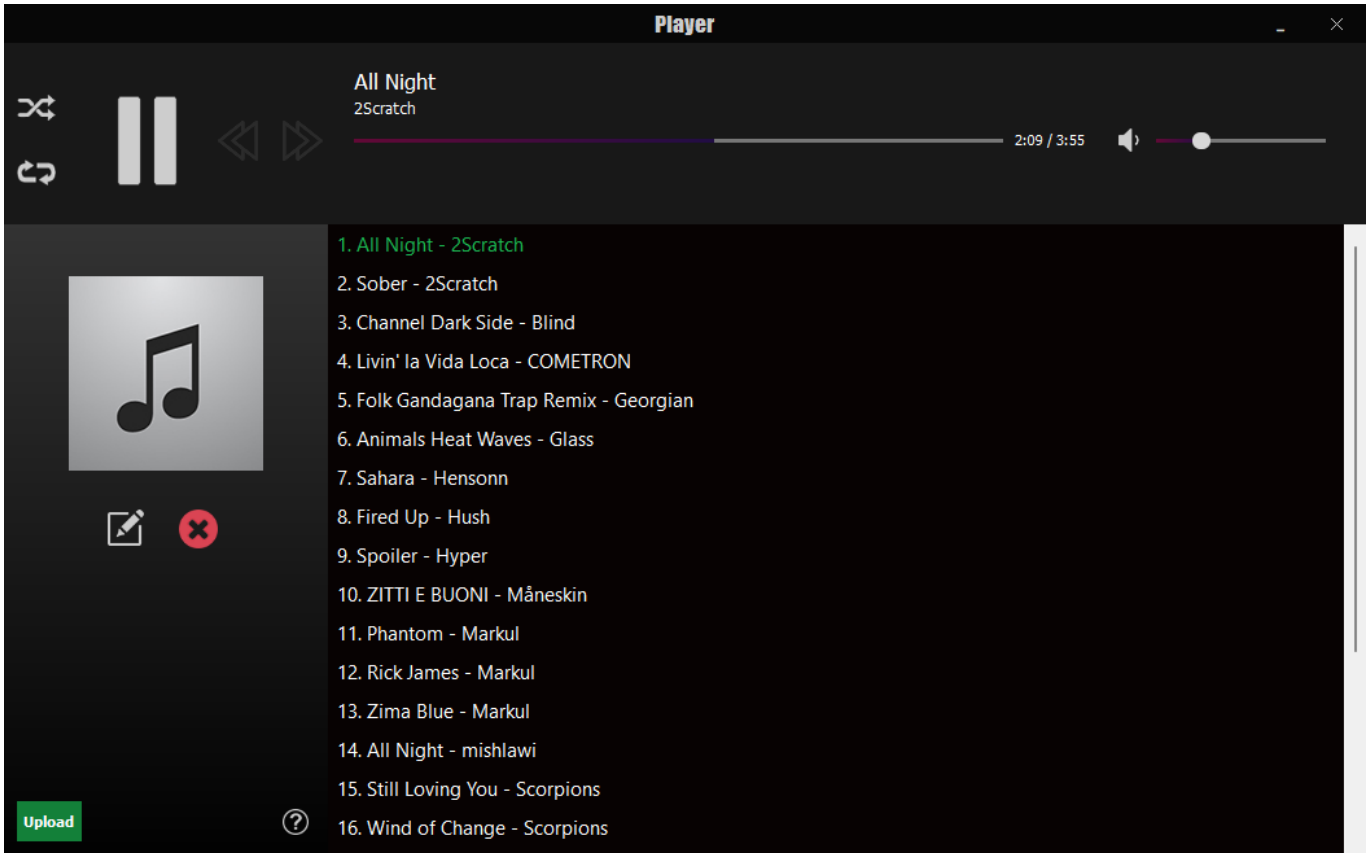


Figura 9 – Music player-ul cu playlist-ul incarcat

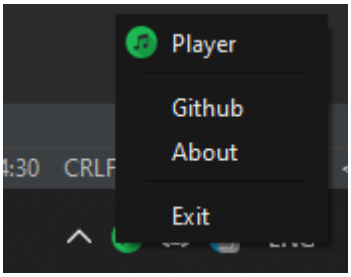


Figura 10 – Tray menu aplicatiei

In figura 10 putem observa ca daca am apasat sa inchidem fereastra aplicatie (butonul „X” din dreapta-sus), aplicatia lucreaza in background, iar icon-ul aplicatiei dispare din menu bar-ul sistemei de operare. Pentru a inchide aplicatia trebuie sa apasam click-drept pe icon-ul aplicatiei din tray menu si de ales optiunea „Exit”.

Concluzii

In concluzia, dupa proiectarea aplicatiei de acest tip, am capatat abilitati care vor fi un avantaj pe viitor, de exemplu la angajare sau ceva de genu. De asemenea m-am familirizat cu biblioteca Qt (PyQt5), mai ales cu clasele si metodele ei care au o gama larga de implementare si sunt disponibile pentru toti gratis (open-source).

UML care a fost predat in anul 2 la obiectul AMOO, in sfarsit a fost utilizat undeva, nu doar in laboratoare/teste. De asemenea informatia predata la curs a fost utilizata cu succes la crearea diagramelor pentru aplicatia data.

Prin urmare obiectivele care au fost puse in plan la Practica tehnologica au fost realizate cu succes.

Bibliografie

1. https://en.wikipedia.org/wiki/Python_(programming_language)
2. https://en.wikipedia.org/wiki/Python_Package_Index
3. https://en.wikipedia.org/wiki/Qt_(software)
4. https://en.wikipedia.org/wiki/PyQt
5. https://en.wikipedia.org/wiki/Unified_Modeling_Language

Anexa A

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox, QSystemTrayIcon,
QAction, qApp, QMenu, QFileDialog
from PyQt5.QtMultimedia import QMediaPlayer, QMediaPlaylist, QMediaContent
from PyQt5.QtGui import QPixmap, QIcon, QColor, QDesktopServices
from playerUI import Ui_MainWindow
import upload
from PyQt5.QtCore import QUrl, QTimer, Qt, QPoint, QDir
import os
import sys
import json
import shutil
from mutagen.id3 import ID3

def suppress_qt_warnings():
    os.environ["QT_DEVICE_PIXEL_RATIO"] = "0"
    os.environ["QT_AUTO_SCREEN_SCALE_FACTOR"] = "1"
    os.environ["QT_SCREEN_SCALE_FACTORS"] = "1"
    os.environ["QT_SCALE_FACTOR"] = "1"


class PlayerWindow(QMainWindow):
    def __init__(self):
        super(PlayerWindow, self).__init__()

        # Setup main window
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.setFixedSize(self.width(), self.height())
        self.setWindowTitle(name_window)
        self.setWindowIcon(QIcon('player.ico'))

        # Setup elements Nr.1
        self.first = True
        self.login_show = True
        self.volume = 50
        self.titles = []
        self.artists = []
        self.covers = []
        self.shuffle = False
        self.repeatthis = False
        self.repeatonce = False
        self.changeMode = False
        self.mode = "Normal"
        self.now_sec = 0
        self.currentIndex = 0

        # Read file with songs and settings
        self.row = 0
        self.read_songs_from_json()
        self.settings_read()
        self.checkCover()

        # Setup elements Nr.2
        self.isPlaying = False
        self.ui.musicSlider.setPageStep(0)
        self.valueSlider = 0
        self.newIndex = -1
        self.playlist.setPlaybackMode(3)
        self.ui.listWidget.setCurrentRow(0)

self.ui.imgLabel.setPixmap(QPixmap("img/no_image.jpg").scaled(self.ui.imgLabel.width(
), self.ui.imgLabel.width()))
```

```
        # Check if exist first song in file
        try:
            self.ui.titleLabel.setText(self.titles[self.row])
            self.ui.artistLabel.setText(self.artists[self.row])
            self.player.playlist().setCurrentIndex(self.row)
            self.ui.listWidget.setCurrentRow(self.row)
            first_song = True
        except Exception as e:
            print(e)
            first_song = False
            self.row = 0

        # Volume and duration label
        self.player.setVolume(self.volume)
        self.ui.durationLabel.setText("0:00 / 0:00")
        self.lastVolume = self.volume

        # Connect buttons
        self.ui.playButton.clicked.connect(self.play)
        self.ui.nextButton.clicked.connect(self.next)
        self.ui.prevButton.clicked.connect(self.prev)
        self.ui.shuffleButton.clicked.connect(self.shuffleMode)
        self.ui.repeatThis.clicked.connect(self.repeatThisMode)
        self.ui.uploadButton.clicked.connect(self.upload_btn)
        self.ui.playButton.setIcon(QIcon("play.png"))
        self.ui.volumeButton.clicked.connect(self.mute)
        self.ui.edit_btn.clicked.connect(self.edit_btn)
        self.ui.deleteButton.clicked.connect(self.delete_btn)
        self.ui.aboutButton.clicked.connect(self.aboutButton)
        self.ui.closeButton.clicked.connect(self.closeButton_clicked)
        self.ui.minimizeButton.clicked.connect(self.minimizeButton_clicked)

        # Music slider bar connect
        self.ui.musicSlider.sliderReleased.connect(self.sliderValue)
        self.ui.listWidget.itemClicked.connect(self.changeSong)

        # Volume slider bar connect
        self.ui.volumeSlider.setValue(self.volume)
        self.ui.volumeSlider.actionTriggered.connect(self.setVolume)

        # Setup timer
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.time_hit)
        self.timer.start(int(1000 / 60))

        # Get text from current item
        try:
            self.text_item = self.ui.listWidget.currentItem().text()
        except Exception as e:
            print(e)

        # Check mode
        self.checkMode()
        if os.path.exists('songs'):
            self.read_files_songs()
            self.checkstylebuttons()
        else:
            self.checkstylebuttons()

        # Set color if exist first song
        if first_song:
            self.text_item = self.ui.listWidget.currentItem().text()
            self.ui.listWidget.currentItem().setText("❯ " + self.text_item)
```

```
        self.start = QPoint(0, 0)
        self.pressing = False

        # Tray menu
        self.tray_icon = QSystemTrayIcon(self)
        self.tray_icon.setIcon(QIcon("player.ico"))

        show_action = QAction(QIcon("player.ico"), "Player", self)
        github_action = QAction("Github", self)
        about_action = QAction("About", self)
        exit_action = QAction("Exit", self)
        show_action.triggered.connect(self.open_tray_button)
        github_action.triggered.connect(self.open_github)
        about_action.triggered.connect(self.aboutButton)
        exit_action.triggered.connect(qApp.quit)

        tray_menu = QMenu()
        tray_menu.setStyleSheet("QMenu{\n"
                                "background-color: #181818;\n"
                                "color: #EAE9E9;}\n"
                                "QMenu::item{\n"
                                "}\n"
                                "\n"
                                "QMenu::item:selected{\n"
                                "background: #252525;}\n"
                                "\n"
                                "QMenu::separator{\n"
                                "height: 10px;\n"
                                "margin-left: 10px;\n"
                                "margin-right: 5px;}")
        tray_menu.addAction(show_action)
        tray_menu.addSeparator()
        tray_menu.addAction(github_action)
        tray_menu.addAction(about_action)
        tray_menu.addSeparator()
        tray_menu.addAction(exit_action)

        self.tray_icon.setContextMenu(tray_menu)
        self.tray_icon.activated.connect(self.systemIcon)
        self.tray_icon.show()

    # ----------------------------------------------------------------------------------------
----------------------------------------


    # read songs from songs.json
    def read_songs_from_json(self):
        if not os.path.exists('songs'):
            os.makedirs('songs')
        self.player = QMediaPlayer()
        self.playlist = QMediaPlaylist(self.player)
        try:
            with open("songs.json", "r", encoding="utf-8") as file:
                data = json.load(file)
            self.titles.clear()
            self.artists.clear()
            self.covers.clear()
            for i in data["Songs"]:
                # title
                self.titles.append(i["title"])
                # artist
                self.artists.append(i["artist"])
                # cover
                if i["cover"] == "Undefined":
```

```
                    self.covers.append("no_image.jpg")
                else:
                    self.covers.append(i["cover"])
        except Exception as e:
            print(e)
        self.read_files_songs()

    # read songs from dir
    def read_files_songs(self):
        try:
            self.ui.listWidget.clear()
            self.playlist = QMediaPlaylist(self.player)

            count = len(os.listdir("songs"))
            for nr in range(count):
                song_name = str(nr) + ".mp3"

self.playlist.addMedia(QMediaContent(QUrl.fromLocalFile(QDir.currentPath() +
"/songs/" + song_name)))
                self.ui.listWidget.addItem(str(nr + 1) + ". " + self.titles[nr] + " -
" + self.artists[nr])
        except Exception as e:
            print(e)

        try:
            self.ui.titleLabel.setText(self.titles[self.row])
            self.ui.artistLabel.setText(self.artists[self.row])
            self.player.setPlaylist(self.playlist)
            self.currentIndex = self.row
            self.player.playlist().setCurrentIndex(self.currentIndex)
            self.ui.listWidget.setCurrentRow(self.currentIndex)
            self.text_item = self.ui.listWidget.currentItem().text()

        except Exception as e:
            print(e)
            try:
                self.row = 0
                self.ui.titleLabel.setText(self.titles[self.row])
                self.ui.artistLabel.setText(self.artists[self.row])
                self.player.setPlaylist(self.playlist)
                self.currentIndex = self.row
                self.player.playlist().setCurrentIndex(self.currentIndex)
                self.ui.listWidget.setCurrentRow(self.currentIndex)
                self.text_item = self.ui.listWidget.currentItem().text()
            except Exception as e:
                print(e)

        self.checkCover()

        try:
            self.player.setVolume(self.volume)
        except Exception as e:
            print(e)
            self.player.setVolume(50)

        try:
            if self.mode == "Shuffle":
                self.playlist.setPlaybackMode(4)
            elif self.mode == "Repeat This":
                self.playlist.setPlaybackMode(1)
            elif self.mode == "Repeat Once":
                self.playlist.setPlaybackMode(0)
            else:
                self.mode = "Normal"
```

```
                self.playlist.setPlaybackMode(3)
        except Exception as e:
            print(e)
            self.playlist.setPlaybackMode(3)

    # Upload button
    def upload_btn(self):
        self.timer.stop()
        completed = False
        try:
            with open("songs.json", "r", encoding="utf-8") as file:
                songs_list = json.load(file)
        except:
            songs_list = {}
            songs_list["Songs"] = []
        window.setEnabled(False)
        if not os.path.exists('songs'):
            os.makedirs('songs')
        nr_of_files = len(os.listdir("songs"))
        try:
            fname = QFileDialog.getOpenFileNames(self, "Open File", "", "MP3 Files
(*.mp3)")
            if not len(fname[0]) == 0:
                self.setWindowTitle(name_window + " | Uploading... 0%")
                self.ui.titleBarInfoLabel.setText("Uploading... 0%")
                QApplication.processEvents()
                nr = len(fname[0])
                for i in range(nr):
                    percent = round((i / nr) * 100)
                    self.setWindowTitle(name_window + " | Uploading... " +
str(percent) + "%")
                    self.ui.titleBarInfoLabel.setText("Uploading... " + str(percent)
+ "%")
                    QApplication.processEvents()
                    path = fname[0][i].split("/")
                    file_name_with_ext = path[-1]
                    file_name = file_name_with_ext.rsplit(".", 1)[0]

                    try:
                        if file_name.__contains__("-"):
                            info_song = file_name.split('-', 1)
                        elif file_name.__contains__(" "):
                            info_song = file_name.split(' ', 1)
                        else:
                            info_song = file_name

                        if len(info_song) == 2:
                            song_name = info_song[1].rstrip().strip()
                            artist = info_song[0].strip().rstrip()
                        elif len(info_song) == 1:
                            song_name = info_song[0].rstrip().strip()
                            artist = ""
                        elif len(info_song) > 2:
                            song_name = file_name.rstrip().strip()
                            artist = ""
                        else:
                            song_name = ""
                            artist = ""
                    except Exception as e:
                        print(e)
                        print("info_song")
                        song_name = ""
                        artist = ""
```

```
                    upload.start(file_name_with_ext, song_name, artist, nr_of_files,
i, nr)
                    while not upload.done:
                        QApplication.processEvents()

                    upload.done = False

                    if str(upload.ui.lineEditName.text()) == "":
                        song_name = "Undefined"
                    else:
                        song_name = str(upload.ui.lineEditName.text())

                    if str(upload.ui.lineEditArtist.text()) == "":
                        artist = "Undefined"
                    else:
                        artist = str(upload.ui.lineEditArtist.text())

                    upload.ui.lineEditName.clear()
                    upload.ui.lineEditArtist.clear()
                    upload.ui.coverLabelInfo.clear()
                    upload.ui.selectedFileInfo.clear()

                    name_of_song = str(nr_of_files) + ".mp3"
                    shutil.copy(fname[0][i], "./songs/" + name_of_song)

                    try:
                        mp3 = ID3("./songs/" + name_of_song)
                        mp3.delete()
                    except Exception as e:
                        print('no ID3 tag')

                    songs_list["Songs"].append({
                        "id": nr_of_files,
                        "title": song_name,
                        "artist": artist,
                        "cover": upload.file_name_final
                    })

                    nr_of_files += 1
                completed = True
        except Exception as e:
            completed = False
            print(e)
        if completed:
            upload.skip_clicked = False
            songs_list["Songs"].sort(key=lambda x: x["id"])
            with open("songs.json", "w", encoding="utf-8") as file:
                json.dump(songs_list, file, indent=4)
            self.read_songs_from_json()
        self.isPlaying = False
        self.setWindowTitle(name_window)
        self.ui.titleBarInfoLabel.setText("")
        self.timer.start()
        window.setEnabled(True)

    # Delete button
    def delete_btn(self):
        id_selected = self.row
        try:
            with open("songs.json", "r", encoding="utf-8") as file:
                songs_list = json.load(file)
            open_file = True
        except:
            print("No songs!")
```

```
            open_file = False

        if open_file:
            last_id = 0
            songs_list_new = {}
            songs_list_new["Songs"] = []

            for song in songs_list["Songs"]:
                if song["id"] == id_selected:
                    os.remove("./songs/" + str(id_selected) + ".mp3")
                    if not song["cover"] == "Undefined":
                        os.remove("./covers/" + song["cover"])
                else:
                    os.rename("./songs/" + str(song["id"]) + ".mp3", "./songs/" +
str(last_id) + ".mp3")
                    if not song["cover"] == "Undefined":
                        cover_name_with_ex = song["cover"]
                        ext = cover_name_with_ex.split(".")[1]
                        cover_new_name = str(last_id) + "." + ext
                        os.rename("./covers/" + song["cover"], "./covers/" +
cover_new_name)
                        songs_list_new["Songs"].append({
                            "id": last_id,
                            "title": song["title"],
                            "artist": song["artist"],
                            "cover": cover_new_name
                        })
                    else:
                        songs_list_new["Songs"].append({
                            "id": last_id,
                            "title": song["title"],
                            "artist": song["artist"],
                            "cover": song["cover"]
                        })

                    last_id += 1

            with open("songs.json", "w", encoding="utf-8") as file:
                json.dump(songs_list_new, file, indent=4)
            self.isPlaying = False
            self.read_songs_from_json()

    # Edit button
    def edit_btn(self):
        self.timer.stop()
        self.setEnabled(False)
        cancel_edit = False
        id_selected = self.row
        try:
            with open("songs.json", "r", encoding="utf-8") as file:
                songs_list = json.load(file)
            open_file = True
        except:
            print("No songs!")
            open_file = False

        if open_file:
            songs_list_new = {}
            songs_list_new["Songs"] = []

            for song in songs_list["Songs"]:
                if song["id"] == id_selected:
                    upload.edit_btn(song["id"], song["title"], song["artist"],
song["cover"])
```

```
                while not upload.done:
                    QApplication.processEvents()
                upload.done = False
                if upload.cancel_edit:
                    cancel_edit = True
                    upload.cancel_edit = False
                    songs_list_new["Songs"].append({
                        "id": song["id"],
                        "title": song["title"],
                        "artist": song["artist"],
                        "cover": song["cover"]
                    })
                else:

                    if str(upload.ui.lineEditName.text()) == "":
                        song_name = "Undefined"
                    else:
                        song_name = str(upload.ui.lineEditName.text())

                    if str(upload.ui.lineEditArtist.text()) == "":
                        artist = "Undefined"
                    else:
                        artist = str(upload.ui.lineEditArtist.text())

                    songs_list_new["Songs"].append({
                        "id": song["id"],
                        "title": song_name,
                        "artist": artist,
                        "cover": upload.file_name_final
                    })

                upload.ui.lineEditName.clear()
                upload.ui.lineEditArtist.clear()
                upload.ui.coverLabelInfo.clear()
                upload.ui.selectedFileInfo.clear()
                upload.ui.pushButton_Skip.setText("Skip all")
            else:
                songs_list_new["Songs"].append({
                    "id": song["id"],
                    "title": song["title"],
                    "artist": song["artist"],
                    "cover": song["cover"]
                })

        if not cancel_edit:
            with open("songs.json", "w", encoding="utf-8") as file:
                json.dump(songs_list_new, file, indent=4)
            self.isPlaying = False
            self.read_songs_from_json()
            self.timer.start()
            window.setEnabled(True)
        else:
            self.timer.start()
            window.setEnabled(True)

    # Tray menu
    def open_tray_button(self):
        if not self.isVisible():
            self.show()
        else:
            self.activateWindow()

    def open_github(self):
        try:
```

```
            url = QUrl("https://github.com/dani3lz/Music_Player")
            QDesktopServices.openUrl(url)
        except Exception as e:
            print(e)

    def systemIcon(self, reason):
        if reason == QSystemTrayIcon.Trigger:
            if self.windowState() == Qt.WindowMinimized:
                self.setWindowState(Qt.WindowNoState)
            else:
                if not self.isVisible():
                    self.show()
                else:
                    self.activateWindow()

    # Check mouse press event
    def mousePressEvent(self, event):
        self.start = self.mapToGlobal(event.pos())
        self.pressing = True

    # Drag app
    def mouseMoveEvent(self, event):
        if self.pressing and (
                self.ui.titleBarLabel.underMouse() or
self.ui.titleBarInfoLabel.underMouse() or self.ui.titleBarTitle.underMouse()):
            self.end = self.mapToGlobal(event.pos())
            self.movement = self.end - self.start
            self.setGeometry(self.mapToGlobal(self.movement).x(),
                             self.mapToGlobal(self.movement).y(),
                             self.width(),
                             self.height())
            self.start = self.end

    # Minimize App
    def minimizeButton_clicked(self):
        self.showMinimized()

    # Close App
    def closeButton_clicked(self):
        self.hide()

    # Close event in minimized status
    def closeEvent(self, event):
        event.ignore()
        self.hide()

    # Function for About button
    def aboutButton(self):
        try:
            self.show()
            self.msg_about = QMessageBox()
            self.msg_about.setWindowTitle("About")
            self.msg_about.setWindowIcon(QIcon("img/about.ico"))
            self.msg_about.setText("Player<br>"
                                   "Version: 1.0<br>"
                                   "Developer: Daniel Zavorot (dani3lz)<br>"
                                   "Github: <a
href='https://github.com/dani3lz/Music_Player'>https://github.com/dani3lz/Music_Playe
r</a>")
            self.msg_about.show()
        except Exception as e:
            print(e)

    # Mute - function for volume
```

```
    def mute(self):
        if self.volume > 0:
            self.lastVolume = self.volume
            self.volume = 0
            self.ui.volumeSlider.setValue(0)
            self.player.setVolume(0)
        else:
            if self.lastVolume > 0:
                self.volume = self.lastVolume
                self.ui.volumeSlider.setValue(self.volume)
                self.player.setVolume(self.volume)
            else:
                self.ui.volumeSlider.setValue(75)
                self.player.setVolume(75)

    # Convert duration of song to minutes and seconds
    def convertMillis(self, millis):
        seconds = (millis / 1000) % 60
        minutes = (millis / (1000 * 60)) % 60
        return minutes, seconds

    # Volume slider
    def setVolume(self):
        self.volume = self.ui.volumeSlider.value()
        self.player.setVolume(self.volume)

    # Change music using the list
    def changeSong(self):
        self.row = self.ui.listWidget.currentRow()
        self.player.playlist().setCurrentIndex(self.row)
        if not self.isPlaying:
            self.player.play()
            self.ui.playButton.setStyleSheet("background-color: transparent;\n"
                                             "border-image: url(img/pause.png);\n"
                                             "background: none;\n"
                                             "border: none;\n"
                                             "background-repeat: none;")
            self.isPlaying = True

    # Music slider
    def sliderValue(self):
        self.player.setPosition(self.ui.musicSlider.value())

    # Read information about player
    def settings_read(self):
        try:
            with open("settings.json", "r", encoding="utf-8") as f:
                data = json.load(f)
            for i in data["Settings"]:
                self.volume = i["Volume"]
                self.lastVolume = self.volume
                self.row = i["Row"]
                self.mode = i["Mode"]
            self.currentIndex = self.row
        except Exception as e:
            print(e)

    # Check player mode
    def checkMode(self):
        if self.mode == "Shuffle":
            self.shuffleMode()
        elif self.mode == "Repeat This":
            self.repeatThisMode()
        elif self.mode == "Repeat Once":
```

```
            self.repeatthis = True
            self.repeatThisMode()

    # Write current information about player
    def settings_write(self):
        settings_list = {}
        settings_list["Settings"] = []
        settings_list["Settings"].append({
            "Volume": self.volume,
            "Row": self.row,
            "Mode": self.mode
        })
        with open("settings.json", "w", encoding="utf-8") as f:
            json.dump(settings_list, f, indent=4)

    # Timer
    def time_hit(self):
        self.checkStyle()
        self.checkstyleVolume()
        if self.isPlaying:
            self.ui.musicSlider.setMaximum(self.player.duration())
            if not self.ui.musicSlider.isSliderDown():
                self.ui.musicSlider.setValue(self.player.position())
            self.newIndex = self.player.playlist().currentIndex()
            self.checkList()

            song_min, song_sec = self.convertMillis(int(self.player.duration()))
            if song_sec < 10:
                self.song_duration = "{0}:0{1}".format(int(song_min), int(song_sec))
            else:
                self.song_duration = "{0}:{1}".format(int(song_min), int(song_sec))

            now_min, self.now_sec =
self.convertMillis(int(self.ui.musicSlider.value()))
            if self.now_sec < 10:
                self.now_duration = "{0}:0{1}".format(int(now_min),
int(self.now_sec))
            else:
                self.now_duration = "{0}:{1}".format(int(now_min), int(self.now_sec))

            self.ui.durationLabel.setText(str(self.now_duration) + " / " +
str(self.song_duration))

            if self.repeatonce:
                if self.now_duration == self.song_duration:
                    self.isPlaying = False
                    self.ui.playButton.setStyleSheet("background-color:
transparent;\n"
                                                     "border-image:
url(img/play.png);\n"
                                                     "background: none;\n"
                                                     "border: none;\n"
                                                     "background-repeat: none;")
                    self.player.stop()
        self.settings_write()

    # Check cover image
    def checkCover(self):
        try:
            if self.covers[self.currentIndex] == "no_image.jpg":
                self.imgsrc = QPixmap("img/" + self.covers[self.currentIndex])
            else:
                self.imgsrc = QPixmap("covers/" + self.covers[self.currentIndex])
            self.w = self.ui.imgLabel.width()
```

```
            self.h = self.ui.imgLabel.width()
            self.ui.imgLabel.setPixmap(self.imgsrc.scaled(self.w, self.h))
        except Exception as e:
            print(e)

    # Sets the current position in the list
    def checkList(self):
        try:
            if self.currentIndex == self.newIndex:
                pass
            else:
                self.ui.listWidget.item(self.currentIndex).setText(self.text_item)

self.ui.listWidget.item(self.currentIndex).setForeground(QColor("#fff"))

                self.text_item = self.ui.listWidget.item(self.newIndex).text()

self.ui.listWidget.item(self.newIndex).setForeground(QColor("#1DB954"))
                self.ui.listWidget.item(self.newIndex).setText("❯ " + self.text_item)

                self.ui.titleLabel.setText(self.titles[self.newIndex])
                self.ui.artistLabel.setText(self.artists[self.newIndex])

self.ui.listWidget.setCurrentRow(self.player.playlist().currentIndex())
                self.currentIndex = self.newIndex
                self.row = self.newIndex
                self.checkCover()
        except Exception as e:
            print(e)

    # Play button
    def play(self):
        if len(self.titles) > 0:
            if not self.isPlaying:
                self.player.play()
                self.isPlaying = True
                self.newIndex = self.player.playlist().currentIndex()
                self.checkStyle()
            else:
                self.player.pause()
                self.isPlaying = False
                self.checkStyle()

    # Next button
    def next(self):
        if len(self.titles) > 0:
            self.playlist.next()
            self.newIndex = self.player.playlist().currentIndex()
            if not self.isPlaying:
                self.player.play()
                self.isPlaying = True
                self.ui.playButton.setStyleSheet("background-color: transparent;\n"
                                                 "border-image:
url(img/pause.png);\n"
                                                 "background: none;\n"
                                                 "border: none;\n"
                                                 "background-repeat: none;")

    # Previous button
    def prev(self):
        if len(self.titles) > 0:
            if int(self.now_sec) < 10:
                self.playlist.previous()
                self.newIndex = self.player.playlist().currentIndex()
```

```
            else:
                self.player.setPosition(0)
            if not self.isPlaying:
                self.player.play()
                self.isPlaying = True
                self.ui.playButton.setStyleSheet("background-color: transparent;\n"
                                                 "border-image:
url(img/pause.png);\n"
                                                 "background: none;\n"
                                                 "border: none;\n"
                                                 "background-repeat: none;")

    # Repeat This button
    def repeatThisMode(self):
        if not self.repeatthis and not self.repeatonce:
            self.playlist.setPlaybackMode(1)
            self.repeatthis = True
            self.shuffle = False
            self.repeatonce = False
            self.mode = "Repeat This"
            self.checkstylebuttons()
        elif self.repeatthis:
            self.playlist.setPlaybackMode(0)
            self.repeatthis = False
            self.shuffle = False
            self.repeatonce = True
            self.mode = "Repeat Once"
            self.checkstylebuttons()
        else:
            self.playlist.setPlaybackMode(3)
            self.repeatonce = False
            self.mode = "Normal"
            self.checkstylebuttons()

    # Shuffle button
    def shuffleMode(self):
        if not self.shuffle:
            self.playlist.setPlaybackMode(4)
            self.shuffle = True
            self.repeatonce = False
            self.repeatthis = False
            self.mode = "Shuffle"
            self.checkstylebuttons()
        else:
            self.playlist.setPlaybackMode(3)
            self.shuffle = False
            self.mode = "Normal"
            self.checkstylebuttons()

    def checkstylebuttons(self):
        if self.shuffle:
            self.ui.shuffleButton.setStyleSheet("background-color: transparent;\n"
                                                "border-image:
url(img/shuffle_on.png);\n"
                                                "background: none;\n"
                                                "border: none;\n"
                                                "background-repeat: none;")
        else:
            self.ui.shuffleButton.setStyleSheet("background-color: transparent;\n"
                                                "border-image:
url(img/shuffle.png);\n"
                                                "background: none;\n"
                                                "border: none;\n"
                                                "background-repeat: none;")
```

```
        if self.repeatthis and not self.repeatonce:
            self.ui.repeatThis.setStyleSheet("background-color: transparent;\n"
                                             "border-image:
url(img/repeatthis_on.png);\n"
                                             "background: none;\n"
                                             "border: none;\n"
                                             "background-repeat: none;")
        elif not self.repeatthis and self.repeatonce:
            self.ui.repeatThis.setStyleSheet("background-color: transparent;\n"
                                             "border-image:
url(img/repeatonce.png);\n"
                                             "background: none;\n"
                                             "border: none;\n"
                                             "background-repeat: none;")
        else:
            self.ui.repeatThis.setStyleSheet("background-color: transparent;\n"
                                             "border-image:
url(img/repeatthis.png);\n"
                                             "background: none;\n"
                                             "border: none;\n"
                                             "background-repeat: none;")

    def checkStyle(self):
        if self.isEnabled():
            if self.ui.deleteButton.underMouse():
                self.ui.deleteButton.setStyleSheet("background-color: transparent;\n"
                                                   "border-image:
url(img/delete_focus.png);\n"
                                                   "background: none;\n"
                                                   "border: none;\n"
                                                   "background-repeat: none;")
            else:
                self.ui.deleteButton.setStyleSheet("background-color: transparent;\n"
                                                   "border-image:
url(img/delete.png);\n"
                                                   "background: none;\n"
                                                   "border: none;\n"
                                                   "background-repeat: none;")
            if self.ui.edit_btn.underMouse():
                self.ui.edit_btn.setStyleSheet("background-color: transparent;\n"
                                               "border-image:
url(img/edit_focus.png);\n"
                                               "background: none;\n"
                                               "border: none;\n"
                                               "background-repeat: none;")
            else:
                self.ui.edit_btn.setStyleSheet("background-color: transparent;\n"
                                               "border-image:
url(img/edit.png);\n"
                                               "background: none;\n"
                                               "border: none;\n"
                                               "background-repeat: none;")
            if self.ui.aboutButton.underMouse():
                self.ui.aboutButton.setStyleSheet("background-color: transparent;\n"
                                                  "border-image:
url(img/about_focus.png);\n"
                                                  "background: none;\n"
                                                  "border: none;\n"
                                                  "background-repeat: none;")
            else:
                self.ui.aboutButton.setStyleSheet("background-color: transparent;\n"
                                                  "border-image:
url(img/about.png);\n"
```

```
                                                "background: none;\n"
                                                "border: none;\n"
                                                "background-repeat: none;")
            if self.ui.musicSlider.underMouse():
                self.ui.musicSlider.setStyleSheet("QSlider{\n"
                                                "    background-color:
transparent;\n"
                                                "}\n"
                                                "QSlider::groove:horizontal \n"
                                                "{\n"
                                                "    background-color:
transparent;\n"
                                                "    height: 3px;\n"
                                                "}\n"
                                                "QSlider::sub-page:horizontal \n"
                                                "{\n"
                                                "    background-color:
qlineargradient(spread:pad, x1:0, y1:0.494, x2:1, y2:0.5, stop:0 rgba(98, 9, 54,
255), stop:1 rgba(33, 13, 68, 255))\n"
                                                "}\n"
                                                "QSlider::add-page:horizontal \n"
                                                "{\n"
                                                "    background-color: rgb(118,
118, 118);\n"
                                                "}\n"
                                                "QSlider::handle:horizontal \n"
                                                "{\n"
                                                "    background-color: rgb(216,
216, 216);\n"
                                                "    width: 14px;\n"
                                                "    margin: -5px;\n"
                                                "    border-radius: 6px;\n"
                                                "}\n"
                                                "QSlider::handle:horizontal:hover
\n"
                                                "{\n"
                                                "    background-color: rgb(240,
240, 240);\n"
                                                "}")
            else:
                self.ui.musicSlider.setStyleSheet("QSlider{\n"
                                                "    background-color:
transparent;\n"
                                                "}\n"
                                                "QSlider::groove:horizontal \n"
                                                "{\n"
                                                "    background-color:
transparent;\n"
                                                "    height: 3px;\n"
                                                "}\n"
                                                "QSlider::sub-page:horizontal \n"
                                                "{\n"
                                                "    background-color:
qlineargradient(spread:pad, x1:0, y1:0.494, x2:1, y2:0.5, stop:0 rgba(98, 9, 54,
255), stop:1 rgba(33, 13, 68, 255))\n"
                                                "}\n"
                                                "QSlider::add-page:horizontal \n"
                                                "{\n"
                                                "    background-color: rgb(118,
118, 118);\n"
                                                "}\n"
                                                "QSlider::handle:horizontal \n"
                                                "{\n"
                                                "    background-color:
```

```
transparent;\n"
                                                    "    width: 14px;\n"
                                                    "    margin: -5px;\n"
                                                    "    border-radius: 6px;\n"
                                                    "}\n"
                                                    "QSlider::handle:horizontal:hover
\n"
                                                    "{\n"
                                                    "    background-color: rgb(240,
240, 240);\n"
                                                    "}")

            if self.ui.playButton.underMouse():
                if not self.isPlaying:
                    self.ui.playButton.setStyleSheet("background-color:
transparent;\n"
                                                     "border-image:
url(img/play_focus.png);\n"
                                                     "background: none;\n"
                                                     "border: none;\n"
                                                     "background-repeat: none;")
                else:
                    self.ui.playButton.setStyleSheet("background-color:
transparent;\n"
                                                     "border-image:
url(img/pause_focus.png);\n"
                                                     "background: none;\n"
                                                     "border: none;\n"
                                                     "background-repeat: none;")
            else:
                if not self.isPlaying:
                    self.ui.playButton.setStyleSheet("background-color:
transparent;\n"
                                                     "border-image:
url(img/play.png);\n"
                                                     "background: none;\n"
                                                     "border: none;\n"
                                                     "background-repeat: none;")
                else:
                    self.ui.playButton.setStyleSheet("background-color:
transparent;\n"
                                                     "border-image:
url(img/pause.png);\n"
                                                     "background: none;\n"
                                                     "border: none;\n"
                                                     "background-repeat: none;")

            if self.ui.nextButton.underMouse():
                self.ui.nextButton.setStyleSheet("background-color: transparent;\n"
                                                 "border-image:
url(img/next_focus.png);\n"
                                                 "background: none;\n"
                                                 "border: none;\n"
                                                 "background-repeat: none;")
            else:
                self.ui.nextButton.setStyleSheet("background-color: transparent;\n"
                                                 "border-image: url(img/next.png);\n"
                                                 "background: none;\n"
                                                 "border: none;\n"
                                                 "background-repeat: none;")

            if self.ui.prevButton.underMouse():
                self.ui.prevButton.setStyleSheet("background-color: transparent;\n"
                                                 "border-image:
```

```
url(img/prev_focus.png);\n"
                                                "background: none;\n"
                                                "border: none;\n"
                                                "background-repeat: none;")
            else:
                self.ui.prevButton.setStyleSheet("background-color: transparent;\n"
                                                "border-image: url(img/prev.png);\n"
                                                "background: none;\n"
                                                "border: none;\n"
                                                "background-repeat: none;")

    def checkstyleVolume(self):
        if self.isEnabled():
            if self.ui.volumeButton.underMouse():
                if self.ui.volumeSlider.value() == 0:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/mute_focus.png);\n"
                                                       "background: none;\n"
                                                       "border: none;\n"
                                                       "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 0 and
self.ui.volumeSlider.value() <= 30:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/low_focus.png);\n"
                                                       "background: none;\n"
                                                       "border: none;\n"
                                                       "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 30 and
self.ui.volumeSlider.value() <= 70:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/medium_focus.png);\n"
                                                       "background: none;\n"
                                                       "border: none;\n"
                                                       "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 70:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/max_focus.png);\n"
                                                       "background: none;\n"
                                                       "border: none;\n"
                                                       "background-repeat: none;")

            else:
                if self.ui.volumeSlider.value() == 0:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/mute.png);\n"
                                                       "background: none;\n"
                                                       "border: none;\n"
                                                       "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 0 and
self.ui.volumeSlider.value() <= 30:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                       "border-image:
url(img/low.png);\n"
```

```
                                                        "background: none;\n"
                                                        "border: none;\n"
                                                        "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 30 and
self.ui.volumeSlider.value() <= 70:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                        "border-image:
url(img/medium.png);\n"
                                                        "background: none;\n"
                                                        "border: none;\n"
                                                        "background-repeat: none;")
                elif self.ui.volumeSlider.value() > 70:
                    self.ui.volumeButton.setStyleSheet("background-color:
transparent;\n"
                                                        "border-image:
url(img/max.png);\n"
                                                        "background: none;\n"
                                                        "border: none;\n"
                                                        "background-repeat: none;")


# ------------------------------------------------------------------------------------
----------------------------------

if __name__ == "__main__":
    suppress_qt_warnings()
    app = QApplication([])
    name_window = "Player"
    window = PlayerWindow()
    window.show()
    upload = upload.UploadWindow()
    sys.exit(app.exec())
```

File-ul “player.py”

```
from PyQt5.QtWidgets import QMainWindow, QFileDialog
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import Qt
from uploadUI import Ui_Form
import os
import shutil


class UploadWindow(QMainWindow):
    def __init__(self):
        super(UploadWindow, self).__init__()

        # Setup Upload window
        self.ui = Ui_Form()
        self.ui.setupUi(self)
        self.setWindowTitle("Upload")
        self.setWindowIcon(QIcon('linux_player.ico'))
        self.setFixedSize(self.width(), self.height())
        self.setWindowFlags(Qt.WindowStaysOnTopHint)
        self.setWindowIcon(QIcon('player.ico'))

        # Var
        self.file_name_final = "Undefined"
        self.skip_clicked = False
        self.done = False
        self.cancel_edit = False
```

```
        # Connect button
        self.ui.pushButton_Ok.clicked.connect(self.finish)
        self.ui.pushButton_Cover.clicked.connect(self.select_cover)
        self.ui.pushButton_Skip.clicked.connect(self.skip_btn)

    def edit_btn(self, id_song, title, artist, cover):
        self.setWindowTitle("Edit")
        self.ui.pushButton_Skip.setText("Cancel")
        self.ui.pushButton_Ok.setText("Ok")
        self.nr = id_song
        file_name = str(id_song) + ".mp3"
        self.ui.selectedFileInfo.setText(file_name)
        self.ui.lineEditName.setText(title)
        self.ui.lineEditArtist.setText(artist)
        self.ui.coverLabelInfo.setText(cover)
        self.show()

    def start(self, file_name, song_name, artist, nr_of_files, current_nr, end_nr):
        if self.skip_clicked:
            self.ui.lineEditName.setText(song_name)
            self.ui.lineEditArtist.setText(artist)
            self.file_name_final = "Undefined"
            self.done = True
        else:
            current_nr += 1
            self.setWindowTitle("Upload " + str(current_nr) + " / " + str(end_nr))
            self.ui.selectedFileInfo.setText(file_name)
            self.ui.lineEditName.setText(song_name)
            self.ui.lineEditArtist.setText(artist)
            self.file_name_final = "Undefined"
            self.nr = nr_of_files
            if current_nr == end_nr:
                self.ui.pushButton_Ok.setText("Ok")
            else:
                self.ui.pushButton_Ok.setText("Next")
            self.show()

    def finish(self):
        self.hide()
        self.done = True

    def skip_btn(self):
        if self.windowTitle() == "Edit":
            self.cancel_edit = True
            self.done = True
            self.hide()
        else:
            self.skip_clicked = True
            self.done = True
            self.hide()

    def select_cover(self):
        final = "Undefined"
        if not os.path.exists('covers'):
            os.makedirs('covers')
        try:
            fname = QFileDialog.getOpenFileName(self, "Open File", "", "Images (*.png
*.xpm *.jpg)")
            if fname:
                self.ui.coverLabelInfo.setText(fname[0])
                path = fname[0].split("/")
                file_name = path[-1]
                info = file_name.split(".")
                extension = info[-1]
```

```
                final = str(self.nr) + "." + str(extension)
                shutil.copy(fname[0], "./covers/" + final)
        except Exception as e:
            print(e)
        self.file_name_final = final

    def closeEvent(self, event):
        if self.windowTitle() == "Edit":
            self.cancel_edit = True
        self.done = True
        event.accept()
```

File-ul “upload.py”

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1051, 655)
        MainWindow.setFocusPolicy(QtCore.Qt.NoFocus)
        MainWindow.setStyleSheet("")
        MainWindow.setWindowFlag(QtCore.Qt.FramelessWindowHint)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.titleBarLabel = QtWidgets.QLabel(self.centralwidget)
        self.titleBarLabel.setGeometry(QtCore.QRect(0, 0, 963, 30))
        self.titleBarLabel.setStyleSheet("background: #070707;\n"
                                         "color: #CDCDCD;")
        self.titleBarLabel.setText("")
        self.titleBarLabel.setObjectName("titleBarLabel")

        self.titleBarTitle = QtWidgets.QLabel(self.centralwidget)
        self.titleBarTitle.setGeometry(QtCore.QRect(88, 0, 875, 30))
        self.titleBarTitle.setAlignment(QtCore.Qt.AlignCenter)
        font = QtGui.QFont("Impact")
        font.setPointSize(13)
        self.titleBarTitle.setFont(font)
        self.titleBarTitle.setStyleSheet("background: transparent;\n"
                                         "color: #CDCDCD;")
        self.titleBarTitle.setText("Player")
        self.titleBarTitle.setObjectName("titleBarTitle")

        self.titleBarInfoLabel = QtWidgets.QLabel(self.centralwidget)
        self.titleBarInfoLabel.setGeometry(QtCore.QRect(20, 0, 963, 30))
        font = QtGui.QFont("Courier")
        font.setPointSize(13)
        self.titleBarInfoLabel.setFont(font)
        self.titleBarInfoLabel.setStyleSheet("background: transparent;\n"
                                         "color: #CDCDCD;")
        self.titleBarInfoLabel.setText("")
        self.titleBarInfoLabel.setObjectName("titleBarInfoLabel")

        self.closeButton = QtWidgets.QPushButton(MainWindow)
        self.closeButton.setGeometry(QtCore.QRect(1007, 0, 44, 30))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.closeButton.setFont(font)
        self.closeButton.setStyleSheet("QPushButton\n"
"{\n"
"    background: #070707;\n"
```

```
"    color: #cdcdcd;\n"
"    border-style: solid;\n"
"    border-width: 1px;\n"
"    border-color: transparent;\n"
"}\n"
"\n"
"QPushButton::hover\n"
"{\n"
"    background-color: #E81123;\n"
"    color: #fff;\n"
"}\n"
"\n"
"\n"
"QPushButton::pressed\n"
"{\n"
"    background-color: #7D0913;\n"
"    color: #fff;\n"
"}")
        self.closeButton.setObjectName("closeButton")



        self.minimizeButton = QtWidgets.QPushButton(MainWindow)
        self.minimizeButton.setGeometry(QtCore.QRect(963, 0, 44, 30))
        font = QtGui.QFont()
        font.setPointSize(9)
        self.minimizeButton.setFont(font)
        self.minimizeButton.setStyleSheet("QPushButton\n"
"{\n"
"    background: #070707;\n"
"    color: #cdcdcd;\n"
"    border-style: solid;\n"
"    border-width: 1px;\n"
"    border-color: transparent;\n"
"}\n"
"\n"
"QPushButton::hover\n"
"{\n"
"    background-color: #0F0F0F;\n"
"    color: #fff;\n"
"}\n"
"\n"
"\n"
"QPushButton::pressed\n"
"{\n"
"    background-color: #4A4A4A;\n"
"    color: #fff;\n"
"}")
        self.minimizeButton.setObjectName("minimizeButton")


        self.titleLabel = QtWidgets.QLabel(self.centralwidget)
        self.titleLabel.setGeometry(QtCore.QRect(270, 49, 501, 21))
        font = QtGui.QFont()
        font.setPointSize(13)
        self.titleLabel.setFont(font)
        self.titleLabel.setStyleSheet("background: none;\n"
"color: white;")
        self.titleLabel.setText("")
        self.titleLabel.setObjectName("titleLabel")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(-10, 30, 1071, 641))
        self.groupBox.setStyleSheet("background: qlineargradient(spread:pad, x1:1,
y1:0, x2:1, y2:1, stop:0 rgba(65, 65, 65, 255), stop:1 rgba(0, 0, 0, 255));\n"
```

```
"border: none;")
        self.groupBox.setTitle("")
        self.groupBox.setObjectName("groupBox")
        self.prevButton = QtWidgets.QPushButton(self.groupBox)
        self.prevButton.setGeometry(QtCore.QRect(175, 60, 31, 31))
        self.prevButton.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/prev.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.prevButton.setText("")
        self.prevButton.setObjectName("prevButton")
        self.durationLabel = QtWidgets.QLabel(self.groupBox)
        self.durationLabel.setGeometry(QtCore.QRect(790, 64, 61, 21))
        font = QtGui.QFont()
        font.setPointSize(8)
        self.durationLabel.setFont(font)
        self.durationLabel.setStyleSheet("color: white;\n"
"background: none;")
        self.durationLabel.setObjectName("durationLabel")
        self.playButton = QtWidgets.QPushButton(self.groupBox)
        self.playButton.setGeometry(QtCore.QRect(85, 40, 71, 71))
        font = QtGui.QFont()
        font.setBold(False)
        font.setItalic(False)
        font.setUnderline(False)
        font.setWeight(50)
        font.setStrikeOut(False)
        self.playButton.setFont(font)
        self.playButton.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/play.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.playButton.setText("")
        self.playButton.setObjectName("playButton")
        self.nextButton = QtWidgets.QPushButton(self.groupBox)
        self.nextButton.setGeometry(QtCore.QRect(225, 60, 31, 31))
        self.nextButton.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/next.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.nextButton.setText("")
        self.nextButton.setObjectName("nextButton")
        self.volumeSlider = QtWidgets.QSlider(self.groupBox)
        self.volumeSlider.setGeometry(QtCore.QRect(899, 64, 131, 22))
        self.volumeSlider.setStyleSheet("QSlider{\n"
"    background-color: transparent;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::groove:horizontal \n"
"{\n"
"    background-color: transparent;\n"
"    height: 3px;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::sub-page:horizontal \n"
"{\n"
"    background-color: qlineargradient(spread:pad, x1:0, y1:0.494, x2:1, y2:0.5,
```

```
stop:0 rgba(98, 9, 54, 255), stop:1 rgba(33, 13, 68, 255))\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::add-page:horizontal \n"
"{\n"
"    background-color: rgb(118, 118, 118);\n"
"\n"
"\n"
";\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::handle:horizontal \n"
"{\n"
"    background-color: rgb(216, 216, 216);\n"
"    width: 14px;\n"
"    margin: -5px;\n"
"    border-radius: 6px;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::handle:horizontal:hover \n"
"{\n"
"    background-color: rgb(240, 240, 240);\n"
"\n"
"}")
        self.volumeSlider.setMinimum(0)
        self.volumeSlider.setMaximum(100)
        self.volumeSlider.setSingleStep(2)
        self.volumeSlider.setPageStep(10)
        self.volumeSlider.setProperty("value", 50)
        self.volumeSlider.setOrientation(QtCore.Qt.Horizontal)
        self.volumeSlider.setTickPosition(QtWidgets.QSlider.NoTicks)
        self.volumeSlider.setObjectName("volumeSlider")
        self.musicSlider = QtWidgets.QSlider(self.groupBox)
        self.musicSlider.setGeometry(QtCore.QRect(280, 60, 501, 31))
        self.musicSlider.setFocusPolicy(QtCore.Qt.NoFocus)
        self.musicSlider.setStyleSheet("QSlider{\n"
"    background-color: transparent;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::groove:horizontal \n"
"{\n"
"    background-color: transparent;\n"
"    height: 3px;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::sub-page:horizontal \n"
"{\n"
"    background-color: qlineargradient(spread:pad, x1:0, y1:0.494, x2:1, y2:0.5,
stop:0 rgba(98, 9, 54, 255), stop:1 rgba(33, 13, 68, 255))\n"
"\n"
"}\n"
"\n"
"QSlider::add-page:horizontal \n"
"{\n"
```

```
"    background-color: rgb(118, 118, 118);\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::handle:horizontal \n"
"{\n"
"    background-color: transparent;\n"
"    width: 14px;\n"
"    margin: -5px;\n"
"    border-radius: 6px;\n"
"\n"
"}\n"
"\n"
"\n"
"QSlider::handle:horizontal:hover \n"
"{\n"
"    background-color: rgb(240, 240, 240);\n"
"\n"
"}")
        self.musicSlider.setMinimum(0)
        self.musicSlider.setOrientation(QtCore.Qt.Horizontal)
        self.musicSlider.setObjectName("musicSlider")
        self.artistLabel = QtWidgets.QLabel(self.groupBox)
        self.artistLabel.setGeometry(QtCore.QRect(280, 40, 491, 20))
        font = QtGui.QFont()
        font.setPointSize(9)
        self.artistLabel.setFont(font)
        self.artistLabel.setStyleSheet("background: none;\n"
"color: white;")
        self.artistLabel.setText("")
        self.artistLabel.setObjectName("artistLabel")
        self.groupBox_2 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_2.setGeometry(QtCore.QRect(10, 0, 1051, 140))
        self.groupBox_2.setStyleSheet("border: none; background: rgb(24, 24 ,24);")
        self.groupBox_2.setTitle("")
        self.groupBox_2.setObjectName("groupBox_2")
        self.shuffleButton = QtWidgets.QPushButton(self.groupBox_2)
        self.shuffleButton.setGeometry(QtCore.QRect(10, 40, 30, 20))
        self.shuffleButton.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/shuffle.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.shuffleButton.setText("")
        self.shuffleButton.setObjectName("shuffleButton")
        self.repeatThis = QtWidgets.QPushButton(self.groupBox_2)
        self.repeatThis.setGeometry(QtCore.QRect(10, 90, 30, 20))
        self.repeatThis.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/repeatthis.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.repeatThis.setText("")
        self.repeatThis.setObjectName("repeatThis")
        self.volumeButton = QtWidgets.QPushButton(self.groupBox_2)
        self.volumeButton.setGeometry(QtCore.QRect(860, 66, 20, 18))
        self.volumeButton.setStyleSheet("background-color: transparent;\n"
"border-image: url(img/medium.png);\n"
"background: none;\n"
"border: none;\n"
"background-repeat: none;")
        self.volumeButton.setText("")
        self.volumeButton.setObjectName("volumeButton")
```

```
        self.imgLabel = QtWidgets.QLabel(self.groupBox)
        self.imgLabel.setGeometry(QtCore.QRect(60, 180, 150, 150))
        self.imgLabel.setStyleSheet("background: transparent;\n"
"")
        self.deleteButton = QtWidgets.QPushButton(self.groupBox)
        self.deleteButton.setGeometry(QtCore.QRect(145, 360, 30, 30))
        self.deleteButton.setStyleSheet("background-color: transparent;\n"
                                        "border-image: url(img/delete.png);\n"
                                        "background: none;\n"
                                        "border: none;\n"
                                        "background-repeat: none;")
        self.deleteButton.setObjectName("deleteButton")

        self.imgLabel.setText("")
        self.imgLabel.setObjectName("imgLabel")
        self.uploadButton = QtWidgets.QPushButton(self.groupBox)
        self.uploadButton.setGeometry(QtCore.QRect(20, 585, 50, 31))
        self.uploadButton.setStyleSheet("QPushButton\n"
"{\n"
"    background-color: #138039;\n"
"    color: #fff;\n"
"    font-size: 11px;\n"
"    font-weight: bold;\n"
"    border: none;\n"
"    border-radius: 25px;\n"
"    padding: 5px;\n"
"\n"
"}\n"
"\n"
"\n"
"QPushButton::disabled\n"
"{\n"
"    background-color: #5c5c5c;\n"
"\n"
"}\n"
"\n"
"\n"
"QPushButton::pressed\n"
"{\n"
"    background-color: #1DB954;\n"
"\n"
"}\n"
"")
        self.uploadButton.setObjectName("uploadButton")

#----------------------

        self.edit_btn = QtWidgets.QPushButton(self.groupBox)
        self.edit_btn.setGeometry(QtCore.QRect(90, 360, 28, 28))
        self.edit_btn.setStyleSheet("background-color: transparent;\n"
                                    "border-image: url(img/edit.png);\n"
                                    "background: none;\n"
                                    "border: none;\n"
                                    "background-repeat: none;")
        self.edit_btn.setText("")
        self.edit_btn.setObjectName("edit_btn")

        self.aboutButton = QtWidgets.QPushButton(self.groupBox)
        self.aboutButton.setGeometry(QtCore.QRect(225, 591, 20, 20))
        self.aboutButton.setStyleSheet("background-color: transparent;\n"
                                       "border-image: url(img/about.png);\n"
                                       "background: none;\n"
                                       "border: none;\n"
                                       "background-repeat: none;")
```

```
        self.aboutButton.setText("")
        self.aboutButton.setObjectName("aboutButton")
#-----------------------
        self.groupBox_2.raise_()
        self.prevButton.raise_()
        self.durationLabel.raise_()
        self.playButton.raise_()
        self.nextButton.raise_()
        self.volumeSlider.raise_()
        self.musicSlider.raise_()
        self.artistLabel.raise_()
        self.imgLabel.raise_()
        self.titleBarLabel.raise_()
        self.titleBarTitle.raise_()
        self.titleBarInfoLabel.raise_()
        self.uploadButton.raise_()
        self.deleteButton.raise_()
        self.closeButton.raise_()
        self.minimizeButton.raise_()
        self.edit_btn.raise_()
        self.listWidget = QtWidgets.QListWidget(self.centralwidget)
        self.listWidget.setGeometry(QtCore.QRect(250, 170, 801, 485))
        font = QtGui.QFont()
        font.setPointSize(11)
        self.listWidget.setFont(font)
        self.listWidget.setStyleSheet("QListView\n"
"{\n"
"  background-color: #070202;\n"
"  alternate-background-color: transparent;\n"
"   border : none;\n"
"   color: #fff;\n"
"   show-decoration-selected: 1;\n"
"   outline: 0;\n"
"   border: 0px solid #1d1d1d;\n"
"}\n"
"QListView::disabled\n"
"{\n"
"   background-color: #000;\n"
"   color: #212121;\n"
"    border: none;\n"
"}\n"
"QListView::item\n"
"{\n"
"    background-color: transparent;\n"
"    padding: 4px;\n"
"}\n"
"QListView::item:selected:!active\n"
"{\n"
"  color: #1DB954;\n"
"}\n"
"QListView::item:selected\n"
"{\n"
"  color: #1DB954;\n"
"}\n"
"QListView::item:hover {\n"
"    background-color: transparent;\n"
"   border: none;\n"
"   color: #1DB954;\n"
"\n"
"}")
        self.listWidget.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.listWidget.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        self.listWidget.setObjectName("listWidget")
        self.groupBox.raise_()
```

```
        self.titleLabel.raise_()
        self.listWidget.raise_()
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.durationLabel.setText(_translate("MainWindow", "0.00 / 0.00"))
        self.uploadButton.setText(_translate("MainWindow", "Upload"))
        self.closeButton.setText(_translate("MainWindow", "X"))
        self.minimizeButton.setText(_translate("MainWindow", "_"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

File-ul “playerUI.py”

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(434, 215)
        Form.setStyleSheet("background-color: #323232")
        self.nameLabel = QtWidgets.QLabel(Form)
        self.nameLabel.setGeometry(QtCore.QRect(20, 60, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.nameLabel.setFont(font)
        self.nameLabel.setStyleSheet("color:#fff;")
        self.nameLabel.setObjectName("nameLabel")
        self.artistLabel = QtWidgets.QLabel(Form)
        self.artistLabel.setGeometry(QtCore.QRect(20, 100, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.artistLabel.setFont(font)
        self.artistLabel.setStyleSheet("color:#fff;")
        self.artistLabel.setObjectName("artistLabel")
        self.coverLabel = QtWidgets.QLabel(Form)
        self.coverLabel.setGeometry(QtCore.QRect(20, 140, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.coverLabel.setFont(font)
        self.coverLabel.setStyleSheet("color:#fff;")
        self.coverLabel.setObjectName("coverLabel")
        self.pushButton_Cover = QtWidgets.QPushButton(Form)
        self.pushButton_Cover.setGeometry(QtCore.QRect(90, 140, 89, 25))
        self.pushButton_Cover.setObjectName("pushButton_Cover")
        self.pushButton_Cover.setStyleSheet("background: #fff; color:#000;")
        self.coverLabelInfo = QtWidgets.QLabel(Form)
        self.coverLabelInfo.setGeometry(QtCore.QRect(190, 140, 231, 21))
        font = QtGui.QFont()
```

```
        font.setPointSize(11)
        self.coverLabelInfo.setFont(font)
        self.coverLabelInfo.setStyleSheet("color:#fff;")
        self.coverLabelInfo.setText("")
        self.coverLabelInfo.setObjectName("coverLabelInfo")
        self.pushButton_Ok = QtWidgets.QPushButton(Form)
        self.pushButton_Ok.setGeometry(QtCore.QRect(330, 180, 89, 25))
        self.pushButton_Ok.setObjectName("pushButton_Ok")
        self.pushButton_Ok.setStyleSheet("background: #fff; color:#000;")
        self.selectedFileInfo = QtWidgets.QLabel(Form)
        self.selectedFileInfo.setGeometry(QtCore.QRect(20, 10, 401, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.selectedFileInfo.setFont(font)
        self.selectedFileInfo.setStyleSheet("color:#fff;")
        self.selectedFileInfo.setObjectName("selectedFileInfo")
        self.lineEditName = QtWidgets.QLineEdit(Form)
        self.lineEditName.setGeometry(QtCore.QRect(90, 60, 331, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lineEditName.setFont(font)
        self.lineEditName.setStyleSheet("background: #fff; color:#000;")
        self.lineEditName.setObjectName("lineEditName")
        self.lineEditArtist = QtWidgets.QLineEdit(Form)
        self.lineEditArtist.setGeometry(QtCore.QRect(90, 100, 331, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lineEditArtist.setFont(font)
        self.lineEditArtist.setStyleSheet("background: #fff; color:#000;")
        self.lineEditArtist.setObjectName("lineEditArtist")
        self.pushButton_Skip = QtWidgets.QPushButton(Form)
        self.pushButton_Skip.setGeometry(QtCore.QRect(10, 180, 89, 25))
        self.pushButton_Skip.setObjectName("pushButton_Skip")
        self.pushButton_Skip.setStyleSheet("background: #fff; color:#000;")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.nameLabel.setText(_translate("Form", "Name:"))
        self.artistLabel.setText(_translate("Form", "Artist:"))
        self.coverLabel.setText(_translate("Form", "Cover:"))
        self.pushButton_Cover.setText(_translate("Form", "Upload"))
        self.pushButton_Ok.setText(_translate("Form", "Ok"))
        self.selectedFileInfo.setText(_translate("Form", "Name of selected file
mp3"))
        self.pushButton_Skip.setText(_translate("Form", "Skip all"))
```

File-ul “uploadUI.py”