

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и кибербезопасности  
Высшая школа программной инженерии

**Курсовая работа**  
по дисциплине «Конструирование программного обеспечения»

Выполнили  
студенты гр. 5130904/10105

Батушева О. В.  
Кривошапкина Р. И.  
Тимофеев И. В.

Руководитель

Иванов А. С.

Санкт-Петербург  
2024 г

## Оглавление

Определение проблемы.....	2
Выработка требований.....	2
Разработка архитектуры и детальное проектирование.....	3
Кодирование и отладка .....	4
Unit тестирование .....	6
Интеграционное тестирование .....	8
Сборка и запуск .....	10
Вывод.....	10

## **Определение проблемы**

Недоступность простых и бесплатных инструментов для визуализации аудио дорожек затрудняет работу с аудио для пользователей без технических навыков. Существующие решения часто требуют значительных финансовых затрат и не поддерживают мобильные платформы. Отсутствие интеграции с популярными мессенджерами затрудняет быстрый обмен визуализированными аудио дорожками.

## **Выработка требований**

### **1. Основные функции:**

- Обработка аудио файлов в формате MP3.
- Создание визуализации аудио дорожки в виде анимированного видео.
- Отправка сгенерированного видео пользователю в ответ на аудиофайл.

### **2. Функции взаимодействия с пользователем:**

- Команда /start для приветствия и краткого описания функциональности бота.
- Автоматическое распознавание и обработка аудио файлов, отправленных в чат.

### **3. Сообщения пользователю:**

- Информирование пользователя о неподдерживаемом формате аудио файлов (только MP3).
- Подтверждение успешной загрузки аудиофайла.
- Отправка визуализированного видео пользователю.

### **4. Обработка файлов:**

- Сохранение загруженных аудиофайлов на сервере.
- Генерация визуализации с помощью ffmpeg.
- Удаление временных файлов после отправки видео пользователю для оптимизации использования дискового пространства.

### **5. Производительность и устойчивость:**

- Поддержка асинхронной загрузки и обработки файлов для обеспечения быстрой реакции бота.
- Обработка ошибок при неправильном формате файла и предоставление пользователю соответствующего сообщения.

### **6. Интерфейс и удобство использования:**

- Простой и интуитивно понятный интерфейс взаимодействия с ботом.

- Минимальные требования к действиям пользователя: отправка аудиофайла и ожидание результата.

## Разработка архитектуры и детальное проектирование

Основной модуль (main.py): Содержит основной код бота и определяет обработчики сообщений.

Функция создания визуализации (create\_visualization): Использует ffmpeg для генерации видео из аудиофайла.

Функция сжатия видео (compress\_video): Использует библиотеку moviepy для сжатия видео.

Обработчики сообщений: Используют библиотеку aiogram для взаимодействия с Telegram API и обработки входящих сообщений и команд.

Тесты (tests/test\_main.py): Содержат unit-тесты и интеграционные тесты для проверки функциональности.

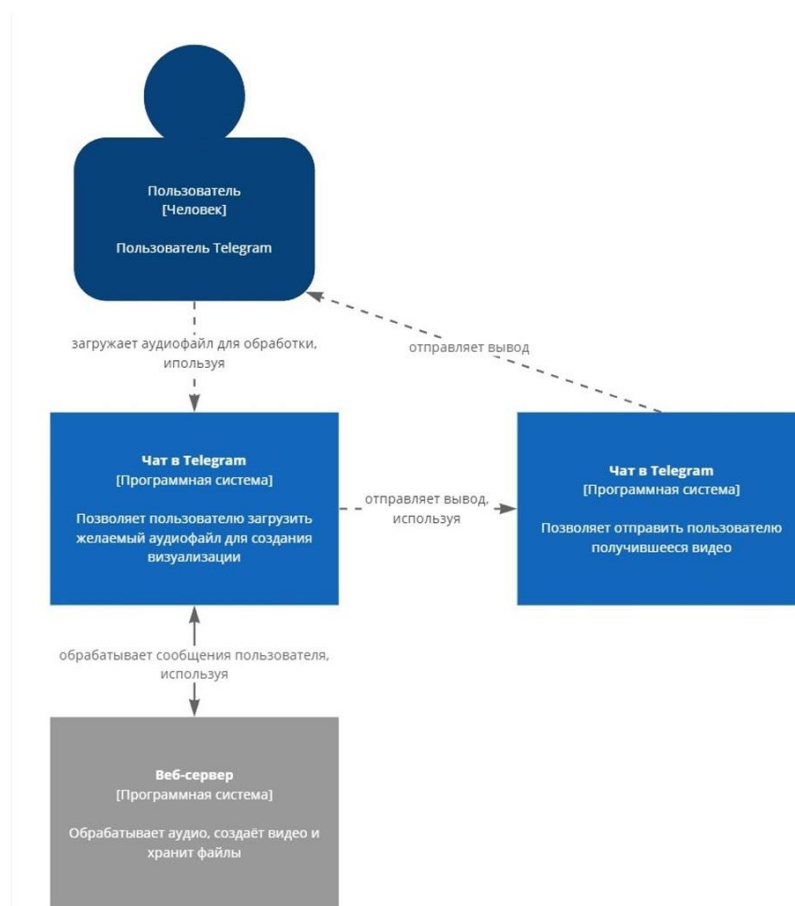


Рисунок 1. Контекстная диаграмма системы

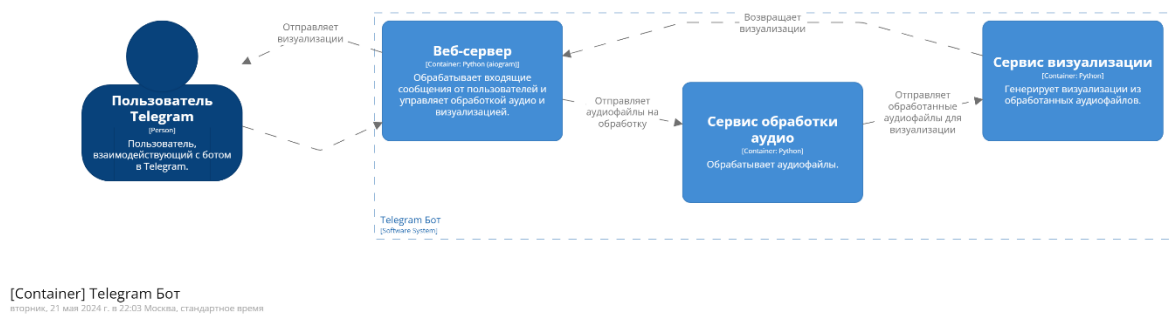


Рисунок 2. Схема контейнера

## Кодирование и отладка

Бот был реализован на Python с использованием библиотек `aiogram` для взаимодействия с Telegram API и `moviepy` для обработки видео. Основные функции были реализованы и протестированы локально.

Исходный код бота:

```

import os
import subprocess
from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher
from aiogram.utils import executor
from moviepy.editor import VideoFileClip

TOKEN = '5804494344:AAEYEObmddT0JQxgUo9MAW_hWVFyJrNhvo'
SAVE_DIR = 'visualization_videos'

bot = Bot(token=TOKEN)
dp = Dispatcher(bot)

def create_visualization(audio_path, output_path):
    subprocess.run(['ffmpeg', '-i', audio_path, '-filter_complex',
'showwaves=s=1280x720:mode=cline', '-y', output_path])

def compress_video(video_path, compressed_path):
    clip = VideoFileClip(video_path)
  
```

```

clip.write_videofile(compressed_path, codec="libx265", bitrate="2500k") #
bitrate
clip.close()

@dp.message_handler(content_types=types.ContentType.AUDIO)
async def process_audio(message: types.Message):
    if message.audio.mime_type != 'audio/mpeg':
        await message.reply('Извините, я могу обрабатывать только аудиофайлы в
формате MP3.')
        return

    audio_file_path = os.path.join(SAVE_DIR,
f'{message.audio.file_unique_id}.mp3')
    await message.audio.download(destination=audio_file_path)

    visualization_file = os.path.join(SAVE_DIR,
f'{message.audio.file_unique_id}.mp4')
    create_visualization(audio_file_path, visualization_file)

    compressed_video_path = os.path.join(SAVE_DIR,
f'{message.audio.file_unique_id}_compressed.mp4')
    compress_video(visualization_file, compressed_video_path)

    with open(compressed_video_path, 'rb') as video:
        await message.reply_video(video)

    os.remove(compressed_video_path)
    os.remove(audio_file_path)

@dp.message_handler(commands=['start'])
async def start(message: types.Message):
    await message.reply("Привет! Я бот, который может создавать
визуализацию аудио в виде анимированного видео. Просто отправь мне
аудиофайл в формате MP3, и я сделаю визуализацию для него!")

```

```
if __name__ == '__main__':  
    os.makedirs(SAVE_DIR, exist_ok=True)  
    executor.start_polling(dp, skip_updates=True)
```

## Unit тестирование

Тестирование функции `create_visualization` с использованием мока для `subprocess.run`.

Тестирование функции `compress_video` с использованием мока для `VideoFileClip`.

Тестирование обработчика сообщений `process_audio`.

Код unit-тестов:

```
import unittest  
from unittest.mock import patch, MagicMock  
import os  
from main import create_visualization, compress_video, process_audio
```

```
class TestCompressVideo(unittest.TestCase):
```

```
    @patch('main.VideoFileClip')  
    def test_compress_video(self, mock_videofileclip):  
        # Создаем мок объекта VideoFileClip  
        mock_clip = MagicMock()  
        mock_videofileclip.return_value = mock_clip
```

```
        video_path = 'test_video.mp4'  
        compressed_path = 'test_compressed.mp4'
```

```
        # Вызываем тестируемую функцию  
        compress_video(video_path, compressed_path)
```

```
        # Проверяем, что VideoFileClip был вызван с правильными  
аргументами  
        mock_videofileclip.assert_called_once_with(video_path)
```

```

        # Проверяем, что методы write_videofile и close были вызваны на
        # объекте mock_clip
        mock_clip.write_videofile.assert_called_once_with(compressed_path,
        codec="libx265", bitrate="2500k")
        mock_clip.close.assert_called_once()

class TestProcessAudioHandler(unittest.TestCase):

    @patch('os.makedirs')
    @patch('os.remove')
    @patch('builtins.open', new_callable=unittest.mock.mock_open)
    @patch('aiogram.types.Message.reply')
    @patch('aiogram.types.Message.reply_video')
    @patch('aiogram.types.Message.audio.download')
    @patch('main.create_visualization')
    @patch('main.compress_video')
    async def test_process_audio(self, mock_compress_video,
mock_create_visualization, mock_download, mock_reply_video, mock_reply,
mock_open, mock_remove, mock_makedirs):
        message = MagicMock()
        message.audio.mime_type = 'audio/mpeg'
        message.audio.file_unique_id = '12345'
        audio_file_path = os.path.join('visualization_videos', '12345.mp3')
        visualization_file = os.path.join('visualization_videos', '12345.mp4')
        compressed_video_path = os.path.join('visualization_videos',
'12345_compressed.mp4')

        await process_audio(message)

        mock_download.assert_called_once_with(destination=audio_file_path)
        mock_create_visualization.assert_called_once_with(audio_file_path,
visualization_file)
        mock_compress_video.assert_called_once_with(visualization_file,
compressed_video_path)
        mock_reply_video.assert_called_once()

```



```

mock_remove.assert_any_call(audio_file_path)
mock_remove.assert_any_call(compressed_video_path)

@patch('aiogram.types.Message.reply')
async def test_process_audio_invalid_mime_type(self, mock_reply):
    message = MagicMock()
    message.audio.mime_type = 'audio/wav'

    with self.assertRaises(Exception):
        await process_audio(message)

    mock_reply.assert_called_once_with('Извините, я могу обрабатывать
только аудиофайлы в формате MP3.')

if __name__ == '__main__':
    unittest.main()

```

## Интеграционное тестирование

Интеграционные тесты проверяют взаимодействие между различными частями системы:

Проверка полной обработки аудиофайла: загрузка, создание визуализации, сжатие и отправка видео.

Проверка обработки неверного MIME-типа.

Код:

```

import unittest
from unittest.mock import patch, MagicMock
import os
from aiogram import types, Dispatcher, Bot
from aiogram.types import ContentType
from main import dp, bot, process_audio, create_visualization,
compress_video

class TestTelegramBotIntegration(unittest.TestCase):

    @patch('main.create_visualization')

```

```

@patch('main.compress_video')
@patch('aiogram.types.Message.reply_video')
@patch('aiogram.types.Message.reply')
@patch('aiogram.types.Message.audio.download')
@patch('os.remove')
async def test_process_audio(self, mock_remove, mock_download,
mock_reply, mock_reply_video, mock_create_visualization,
mock_compress_video):
    # Создаем мок-объект для сообщения
    message = MagicMock(spec=types.Message)
    message.audio.mime_type = 'audio/mpeg'
    message.audio.file_unique_id = '12345'
    message.audio.download.return_value = 'path/to/downloaded/file.mp3'

    audio_file_path = os.path.join('visualization_videos', '12345.mp3')
    visualization_file = os.path.join('visualization_videos', '12345.mp4')
    compressed_video_path = os.path.join('visualization_videos',
'12345_compressed.mp4')

    # Вызываем функцию обработки аудио
    await process_audio(message)

    # Проверяем, что функции были вызваны с правильными
аргументами
    mock_download.assert_called_once_with(destination=audio_file_path)
    mock_create_visualization.assert_called_once_with(audio_file_path,
visualization_file)
    mock_compress_video.assert_called_once_with(visualization_file,
compressed_video_path)
    mock_reply_video.assert_called_once()
    mock_remove.assert_any_call(audio_file_path)
    mock_remove.assert_any_call(compressed_video_path)

@patch('aiogram.types.Message.reply')
async def test_process_audio_invalid_mime_type(self, mock_reply):
    # Создаем мок-объект для сообщения с неверным MIME-типом
    message = MagicMock(spec=types.Message)

```

```
message.audio.mime_type = 'audio/wav'

with self.assertRaises(Exception): # замените CancelHandler на
Exception
    await process_audio(message)

# Проверяем, что бот отправил сообщение с ошибкой
mock_reply.assert_called_once_with('Извините, я могу обрабатывать
только аудиофайлы в формате MP3.')

if __name__ == '__main__':
    unittest.main()
```

## Сборка и запуск

Сборка: `docker build -t my_telegram_bot .`

Запуск: `docker run -d --name my_telegram_bot_container my_telegram_bot`

Тесты: `docker build -t my_telegram_bot_tests -f Dockerfile.tests .`

`docker run --rm my_telegram_bot_tests`

## Вывод

Мы разработали Telegram-бота и описали все основные этапы разработки от определения проблемы до тестирования и сборки.