федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный университет геодезии и картографии»

(МИИГАиК)

Факультет геоинформатики и информационной безопасности

Специальность 09.03.03 «Прикладная информатика»

Кафедра информатики и геоинформационных технологий

ОТЧЕТ ПО

УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКЕ

ПО ПРОГРАММИРОВАНИЮ

Исполнитель студент группы 2024-ФГиИБ-ПИ-1б

Ласица Анастасия Алексеевна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель производственной практики Лебедев Е.Д. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2025

**Дневник практики**

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Выполняемая работа** | **Отметка о выполнении** | **Примечание** |
| 07.07.2025– 08.07.2025 | Ознакомление с заданиями летней практики. Выбор темы проекта (аудиоплеер). Обзор библиотек Flet и Flet Audio. | Выполнено |  |
| 09.07.2025– 10.07.2025 | Разработка архитектуры аудиоплеера. Изучение работы с Flet Audio для управления воспроизведением. | Выполнено |  |
| 11.07.2025– 12.07.2025 | Начало написания класса AudioPlayerManager. Реализация функций загрузки треков (load\_local\_tracks) и управления воспроизведением (play\_pause\_click). | Выполнено |  |
| 14.07.2025– 15.07.2025 | Разработка логики переключения треков (next\_track, prev\_track) и обработки событий аудио (audio\_state\_changed, audio\_position\_changed). | Выполнено |  |
| 16.07.2025– 17.07.2025 | Реализация эквалайзера (EqualizerAnimation) с анимацией полос. Интеграция с основным плеером. | Выполнено |  |
| 18.07.2025– 19.07.2025 | Добавление функций перемешивания (toggle\_shuffle) и повтора треков (toggle\_repeat). | Выполнено |  |
| 21.07.2025– 22.07.2025 | Тестирование функционала аудиоплеера. Доработка логики эквалайзера и обработчиков событий. Написание модульных тестов. Проведение нефункционального тестирования. | Выполнено |  |
| 23.07.2025– 24.07.2025 | Заполнение таблицы тестов. Оформление отчёта практики. | Выполнено |  |
| 25.07.2025 | Подведение итогов. Написание выводов. Составление списка источников. Сдача дневника и финального отчета преподавателю. | Выполнено |  |

**Оглавление**

Введение4

Глава 1 «Описание структуры проекта»5

Глава 2 «Реализация аудиоплеера на Python» 8

2.1. Модуль EqualizerAnimation8

2.2. Модуль AudioPlayerManager 9

2.3. Интеграция с интерфейсом11

Глава 3 «Тестирование аудиоплеера на Python»15

3.1. Модульное тестирование15

3.2. Нефункциональноетестирование17

Выводы19

Список использованных источников20

**Введение**

Для первого серьезного проекта выбрана тема «Разработка аудиоплеера на Python», поскольку музыка играет важную роль в жизни, а создание собственного аудиоплеера представляется увлекательным и вдохновляющим. Разработка программы, способной не только воспроизводить музыку, но и обладать удобным и привлекательным интерфейсом, вызывает интерес.

Проект предоставляет возможность применить и закрепить навыки, полученные в течение учебного года. Знания в программировании на Python, работа с библиотеками, понимание принципов обработки аудиоданных и создания интерфейсов станут основой для реализации. Основное внимание уделяется использованию библиотеки Flet для создания интуитивно понятного графического интерфейса, а также других библиотек Python для работы с аудио.

Проект способствует развитию навыков решения практических задач, углубленному изучению документации и поиску оригинальных подходов к решению проблем, что окажет положительное влияние на дальнейшую учебу и профессиональный рост.

**Глава 1 «Описание структуры проекта»**

Проект представляет собой аудиоплеер, реализованный на Python с использованием библиотеки **Flet** для создания графического интерфейса и управления воспроизведением аудиофайлов. Система предназначена для воспроизведения локальных MP3-файлов, управления плейлистом, визуализации аудио с помощью эквалайзера и отслеживания статистики прослушиваний. Основной акцент в разработке сделан на:

1. **EqualizerAnimation** – класс для создания анимированного эквалайзера, который визуально отображает динамику воспроизведения аудио. Эквалайзер состоит из нескольких полос, высота которых изменяется синусоидально, создавая эффект анимации.
2. **AudioPlayerManager** – класс, управляющий логикой воспроизведения аудио, включая загрузку треков, управление состоянием плеера (воспроизведение, пауза, остановка), переключение треков, режимы повторения и перемешивания, а также обработку событий, связанных с изменением позиции и состояния аудио.

Система состоит из трех основных модулей:

1. **Графический интерфейс** – реализован с помощью **Flet**, включает вкладки для управления плеером, отображения очереди треков и статистики прослушиваний.
2. **Аудиоплеер** – отвечает за воспроизведение аудиофайлов, управление громкостью, позицией воспроизведения и режимами работы (повтор, перемешивание).
3. **Эквалайзер** – визуальный компонент, который синхронизируется с воспроизведением аудио, обеспечивая динамическую анимацию.

Компоненты взаимодействуют через **AudioPlayerManager**, который связывает логику воспроизведения с интерфейсом, и **EqualizerAnimation**, интегрированный в интерфейс для визуального отображения. Проект рассчитан на запуск в среде Python с установленной библиотекой **Flet** и предполагает наличие папки tracks с MP3-файлами для воспроизведения.

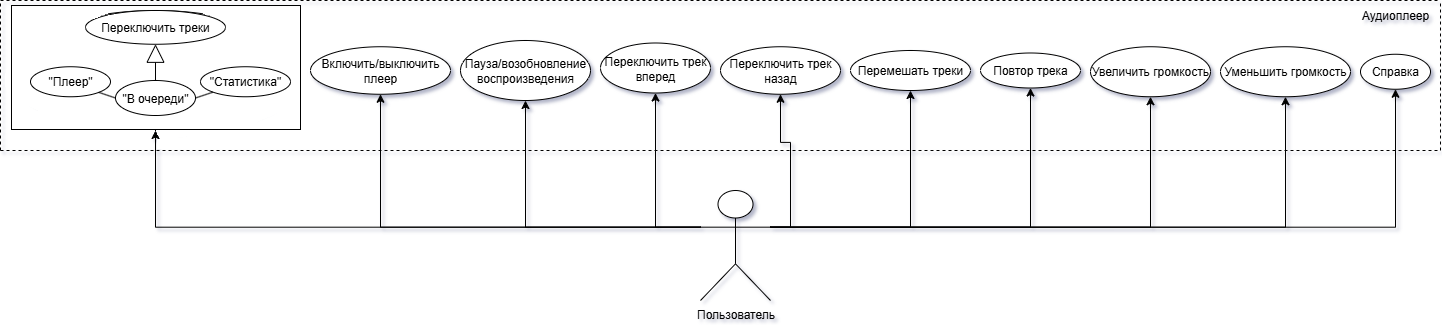


Рис. 1 «Диаграмма вариантов использования»

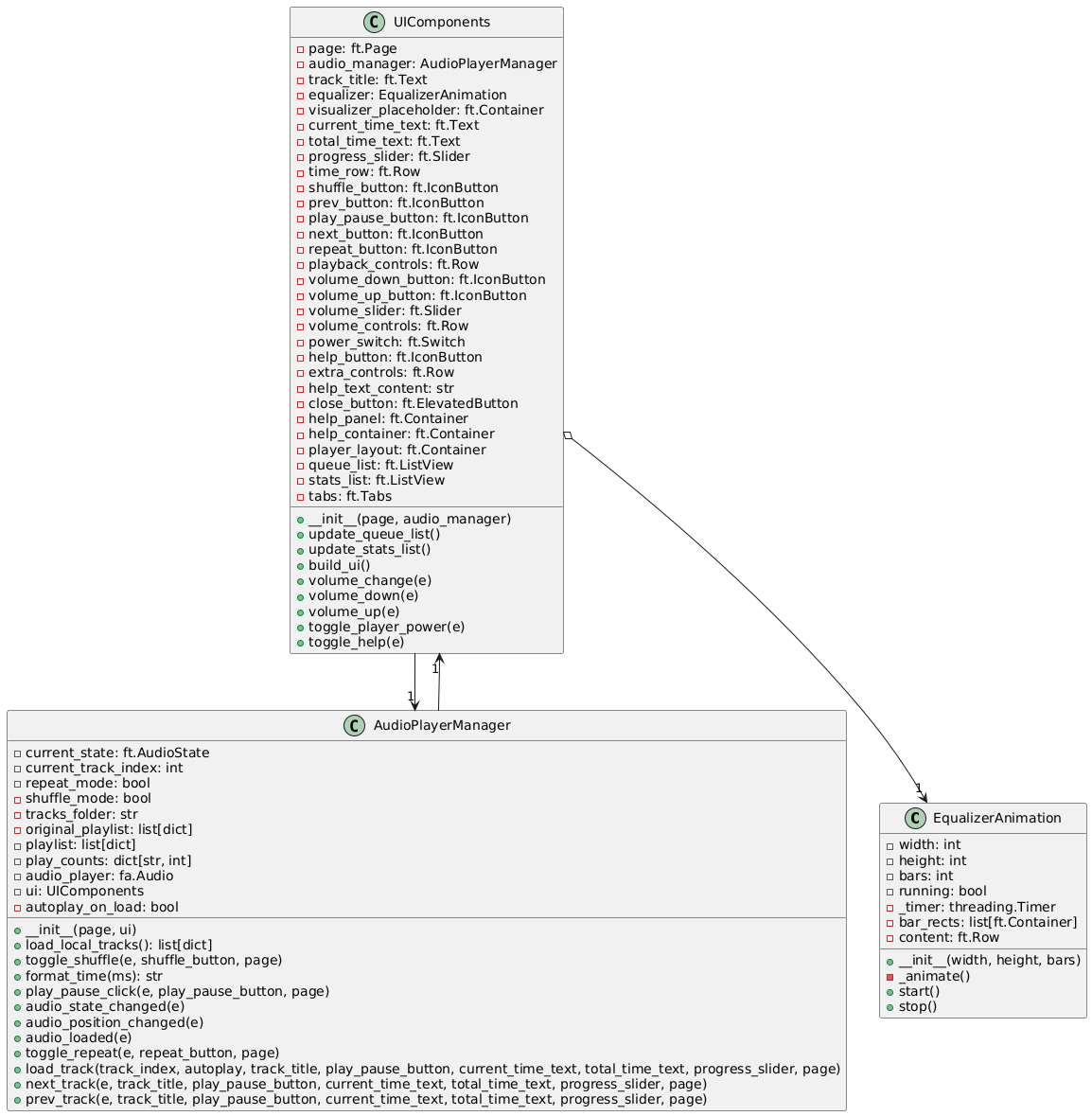


Рис. 2 «Диаграмма классов»

**Глава 2 «Реализация аудиоплеера на Python»**

**2.1. Модуль EqualizerAnimation**

Класс **EqualizerAnimation** отвечает за создание визуального эквалайзера, который отображает анимацию в виде движущихся полос, синхронизированных с воспроизведением треков. Анимация создается с использованием синусоидальной функции для изменения высоты полос. Полосы не анализируют бары треков и не двигаются под бит(а хаотично), потому что так приложение сильно нагружается и перестает работать.

Основные компоненты:

1. Конструктор принимает параметры ширины, высоты и количества полос эквалайзера. Создается список контейнеров **bar\_rects** для каждой полосы с заданными стилями – цвет, радиус закругления, выравнивание.
2. Метод **\_animate** изменяет высоту каждой полосы на основе текущего времени и индекса полосы, используя функцию sin для создания волнообразного эффекта. Анимация обновляется каждые 0.05 секунды с помощью **threading.Timer**.
3. Методы **start** и **stop** запускают и останавливают анимацию, управляя флагом running и таймером.

Построение интерфейса:

1. Эквалайзер размещается в контейнере **ft.Container** с фиксированными размерами – ширина 300, высота 265.
2. Полосы эквалайзера выравниваются по нижнему краю с помощью **ft.Row** и **alignment.bottom\_center.**
3. Цвет полос – **ft.Colors.PURPLE\_ACCENT\_400**, как и остальной дизайн интерфейса.
4. Обновление интерфейса выполняется через метод **update** для плавной анимации.

**2.2. Модуль AudioPlayerManager**

Класс **AudioPlayerManager** управляет всей логикой плеера. Он отвечает за функциональность воспроизведения, паузы, переключения треков, режимов повторения, перемешивания.

Основные компоненты:

* 1. Создается объект **fa.Audio** для воспроизведения аудиофайлов с начальными параметрами: громкость 0.5, автопроигрывание отключено. Загружается список треков из папки tracks и появляется плейлист. Статистика прослушиваний отслеживается через словарь **play\_counts**.
  2. Загрузка треков (**load\_local\_tracks**) – сканируется папка tracks и формируется список словарей с информацией о треках. Загружаются только файлы с расширением .mp3.
  3. Управление воспроизведением:
* **play\_pause\_click** – переключает состояния воспроизведения/паузы, обновляет иконку кнопки и запускает/останавливает эквалайзер.
* **next\_track** и **prev\_track** – переключают треки вперед или назад с учетом режима перемешивания.
* **load\_track** – загружает трек по индексу, обновляет интерфейс и поддерживает автопроигрывание.
  1. Режимы работы:
* **toggle\_shuffle** – включает/выключает режим перемешивания, перестраивает плейлист при включении.
* **toggle\_repeat** – включает/выключает режим повтора текущего трека.
  1. Обработка событий:
* **audio\_state\_changed** – обновляет состояние плеера – воспроизведение, пауза, остановка.
* **audio\_position\_changed** – обновляет позицию воспроизведения, слайдер прогресса и переключает трек при завершении.
* **audio\_loaded** – устанавливает длительность трека и запускает воспроизведение, если включено автопроигрывание.

Построение интерфейса:

1. Все элементы управления (кнопки и слайдеры) взаимодействуют с **AudioPlayerManager** через передачу соответствующих объектов интерфейса (play\_pause\_button, track\_title и т.д.).
2. Обновление интерфейса – через **page.update()** для синхронизации изменений состояния плеера с графическим интерфейсом.
3. События аудиоплеера (изменение позиции, загрузка трека) обрабатываются асинхронно, чтобы не было задержек в интерфейсе.
4. У визуальных элементов цвет – **ft.Colors.PURPLE\_ACCENT\_400** – для активных состояний, и **ft.Colors.GREY** – для неактивных.

**2.3. Интеграция с интерфейсом**

Модули **EqualizerAnimation** и **AudioPlayerManager** интегрированы с интерфейсом, реализованным в классе **UIComponents**. **AudioPlayerManager** передает данные о текущем треке, состоянии воспроизведения и статистике в элементы интерфейса (слайдеры, текстовые поля, списки). Эквалайзер включается и отключается синхронно с воспроизведением через методы *start* и *stop*, вызываемые из **AudioPlayerManager**.

Построение интерфейса:

1. Интерфейс должен быстро реагировать: нажатия кнопок и изменения слайдера сразу обновляют состояние плеера.
2. Списки очереди и статистики (**queue\_list**, **stats\_list**) обновляются динамически при изменении плейлиста или статистики прослушиваний.
3. Визуальные элементы должны быть согласованы по стилю с основным дизайном.
4. Все элементы управления должны быть доступны и интуитивно понятны.

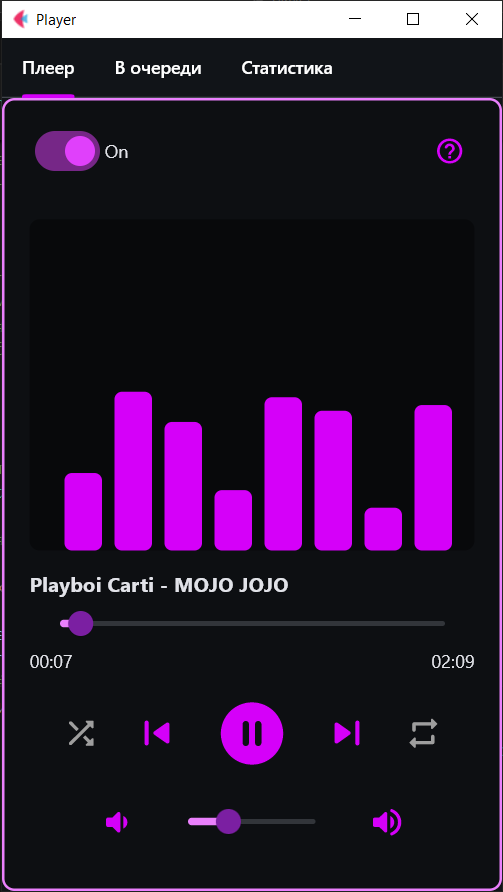


Рис. 3 «Интерфейс плеера»

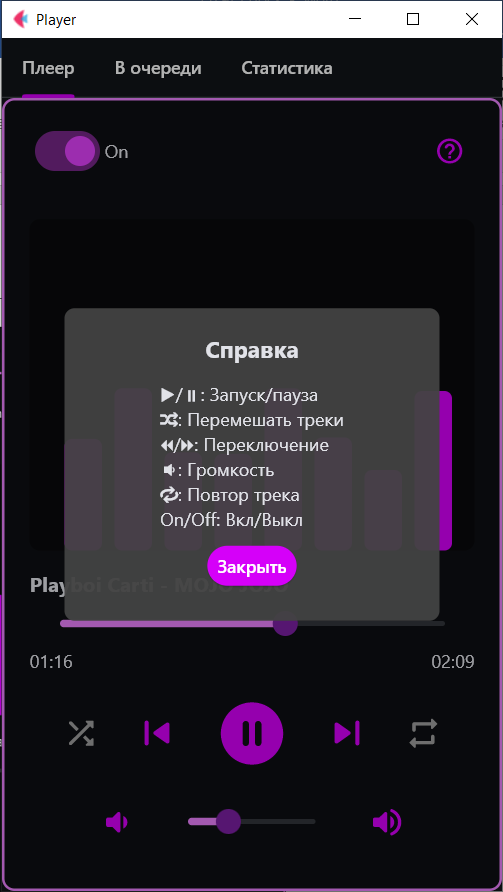


Рис. 4 «Справка»

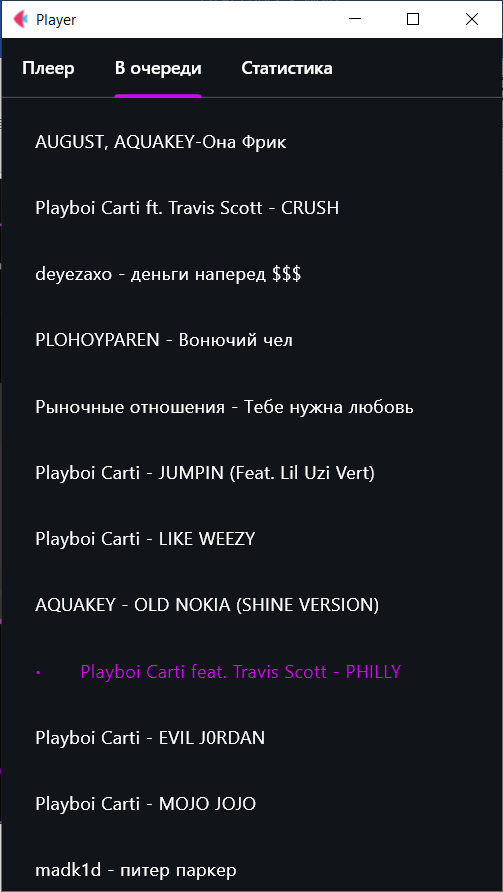


Рис. 5 «Вкладка «В очереди»»

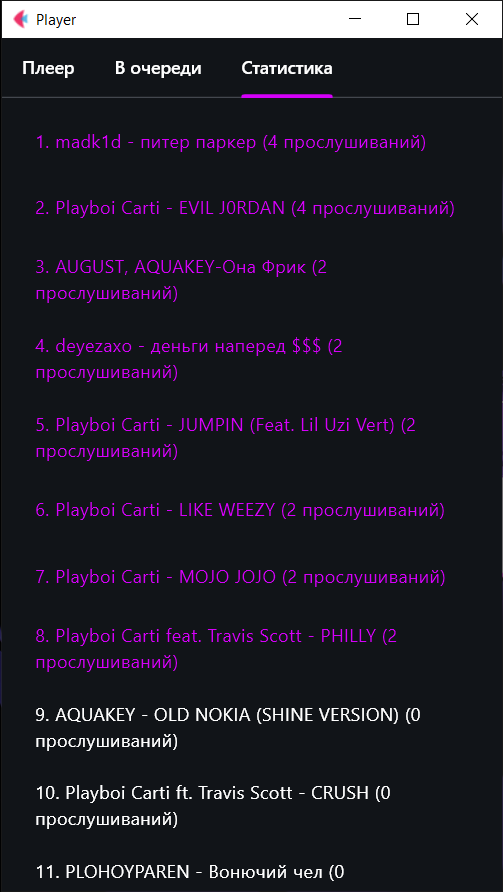


Рис. 6 «Вкладка «Статистика»»

**Глава 3 «Тестирование аудиоплеера на Python»**

Тестирование аудиоплеера проводилось с целью обеспечения его работы без ошибок и удобства использования. Проверялись два типа тестов: модульные (для отдельных функций) и нефункциональные (для оценки скорости).

**3.1. Модульное тестирование**

Тестирование класса **AudioPlayerManager**, управляющего треками, проводилось с использованием библиотеки *unittest*. Проверялись две функции:

* + **load\_local\_tracks** – загружает треки из папки.
  + **toggle\_shuffle** – перемешивает плейлист, сохраняя текущий трек на месте.

Тест 1: Загрузка треков

Проверялась функция **load\_local\_tracks** на способность находить все mp3-файлы в папке. Были созданы два тестовых файла (track1.mp3, track2.mp3), после чего подтверждалось, что плеер их обнаруживает.

Листинг 1. Модульный тест № 1



Тест 2: Перемешивание плейлиста

Проверялась функция **toggle\_shuffle** на корректное изменение порядка треков при сохранении позиции текущего трека. Был сформирован плейлист из трёх треков, после чего подтверждалось, что при включении перемешивания порядок треков изменяется, но текущий трек остаётся на своей позиции.

Листинг 2. Модульный тест № 2



* 1. **. Нефункциональное тестирование**

Проводилась проверка скорости переключения треков и стабильности работы анимации эквалайзера при длительном воспроизведении музыки.

Тест 3: Скорость переключения треков

Выполнялось последовательное переключение 20 треков с проверкой, чтобы каждое переключение занимало менее 0.5 секунды.

Тест 4: Анимация эквалайзера

Воспроизведение треков осуществлялось в течение 10 минут с наблюдением за анимацией эквалайзера для подтверждения отсутствия замедлений.

Таблица 2 «Результаты тестирования»

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Вид тестирования** | **Описание теста** | **Входные данные** | **Ожидаемый результат** | **Фактический результат** | **Статус** | **Кто проводил** |
| 1 | Модульное тестирование | Функция load\_local\_tracks | Папка с файлами: track1.mp3, track2.mp3 | Плеер загружает два трека | Плеер загрузил два трека | Пройдено | Ласица А.А. |
| 2 | Модульное тестирование | Функция toggle\_shuffle | Плейлист из трёх треков, включено перемешивание | Треки перемешаны, текущий трек на месте | Треки перемешаны, текущий трек на месте | Пройдено | Никитинская Т.Г. |
| 3 | Нефункциональное тестирование | Скорость переключения треков | Переключение 20 треков | Переключение за < 0.5 сек | Скорость очень маленкая и незаметная | Пройдено | Ласица А.А. |
| 4 | Нефункциональное тестирование | Анимация эквалайзера | Воспроизведение 10 минут | Анимация не зависает | Анимация не зависает | Пройдено | Никитинская Т.Г. |

Тесты прошли успешно. Плеер работает нормально, ничего не ломается.

**Выводы**

На летней практике был разработан аудиоплеер на Python с использованием библиотеки Flet. Этот проект стал первым крупным опытом, позволившим применить знания, полученные за год учебы. Плеер поддерживает воспроизведение mp3-файлов, управление плейлистом, отображение анимации эквалайзера и ведение статистики прослушиваний.

Создан класс AudioPlayerManager, отвечающий за воспроизведение, паузу, переключение треков, а также за режимы повтора и перемешивания. Все функции работают стабильно.

Класс EqualizerAnimation формирует анимированные полосы, которые отображаются во время воспроизведения музыки. Для предотвращения замедления приложения анимация не синхронизирована с ритмом, но выглядит эффектно.

Разработан удобный интерфейс с тремя вкладками: плеер, список треков и статистика. Выбран фиолетовый цвет для оформления, обеспечивающий яркий и привлекательный дизайн.

Проведено тестирование загрузки треков и скорости их переключения. Все тесты завершены успешно.

В процессе работы изучена документация Flet, освоены принципы обработки асинхронных событий и применения многопоточности для анимации. Также получены навыки тестирования кода и подготовки отчета.

Проект оказался интересным и полезным, позволив освоить интеграцию плеера с интерфейсом, создание анимации и решение задач для обеспечения быстрой и стабильной работы. Хотя эквалайзер пока не реагирует на ритм музыки, плеер получился удобным и эстетичным. В будущем возможно добавление поддержки других аудиоформатов или создание эквалайзера, реагирующего на ритм.

**Список использованных источников**

1. Документация библиотеки Flet [Электронный ресурс] URL: <https://flet.dev/docs/>  (дата обращения 24.07.2025)
2. Simple Music Player in Python - Flet Tutorial [Электронный ресурс] URL: <https://rutube.ru/video/cd0679923ab63e90103c0909d39752ab/> (дата обращения 24.07.2025)
3. Библиотека Python Flet для UI дизайна приложений [Электронный ресурс] URL: <https://itproger.com/course/python-gui/2> (дата обращения 24.07.2025)
4. Гайд по Flet: приступая к работе. Создание приложений на Python [Электронный ресурс] URL: <https://www.ixbt.com/live/sw/gayd-po-flet-pristua.html> (дата обращения 24.07.2025)
5. Колесинский, М. В. Flet: разработка кроссплатформенных приложений на Python / М. В. Колесинский // Цифровые, компьютерные и информационные технологии в науке и образовании: Сборник статей II Межрегиональной научно-практической конференции с международным участием, Брянск, 14–15 ноября 2024 года. – Брянск: Брянский государственный университет им. акад. И.Г. Петровского, 2025. – С. 306-308.
6. Куликов, Д. А. Программирование на Python с использованием библиотеки Flet / Д. А. Куликов, О. В. Осипов // Информационные технологии. Радиоэлектроника. Телекоммуникации (ITRT- 2024): Материалы XI Международной заочной научно-технической конференции, Тольятти, 29 ноября 2024 года. – Тольятти: Издательско-полиграфический центр ПВГУС, 2025. – С. 26-31.