Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательноe учреждение высшего образования  
**«Вятский государственный университет»**  
(ВятГУ)

**Отчет  
по учебной практике**

|  |  |
| --- | --- |
| Ердяков Роман Александрович | |
| *(Ф.И.О. обучающегося)* | |
| 09.03.02 Информационные системы и технологии, 02 Информационные системы и технологии управления технологическими процессами в промышленности | |
| *(направление подготовки (специальность), направленность (профиль))* | |
|  | |
| Место прохождения практики | ФГБОУ ВО «Вятский государственный университет», |
|  | *(наименование организации, структурного подразделения организации)* |
| кафедра систем автоматизации управления | |
| *(наименование организации, структурного подразделения организации)* | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Итоговая оценка: |  | | | | |
| Руководитель практики от ‍университета |  |  |  |  |  |
|  | *(дата)* |  | *(подпись)* |  | *(Ф.И.О.)* |

Киров, 2025 г.

### Реферат

Ердяков Р.А. Разработка настольного приложения «**Книжная коллекция**»: ТПЖА.090302.446 ПЗ: Отчёт по учебной практике / ВятГУ, каф. САУ; рук. А.А. Махмуров. – Киров, 2025. ПЗ 39 с., 21 рис., 6 источников, 6 прил.

ПРИЛОЖЕНИЕ, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС, PYTHON, TKINTER, ИНФОРМАЦИОННАЯ СИСТЕМА, БАЗА ДАННЫХ

Объект исследования и разработки – процесс создания приложения для отображения коллекции сущностей (книг и аудиокниг) с использованием принципов объектно-ориентированного программирования, включая интерфейсы и наследование.

Цель работы – разработка и реализация приложения с графическим интерфейсом, позволяющего управлять коллекцией сущностей книги и аудиокниги. Книги содержат поля: наименование, автор, стоимость. Аудиокниги дополнительно содержат поля: диктор, длительность. Книги и Аудиокниги наследуют общий базовый класс. Реализовать вычисление средней арифметической и медианной стоимости для выбранных элементов списка.

В ходе работы были выполнены анализ предметной области, проектирование интерфейса, разработка приложения с использованием Python, Tkinter и Sqlite, а также проведено тестирование функциональности.

Результатом работы стало готовое к использованию приложение с главным окном для просмотра коллекции, формами для добавления, редактирования и удаления записей, вычислением средней арифметической и медианной стоимости для выбранных элементов списка.

Содержание

[1 Анализ проблемы и требований, обзор технологий 4](#_Toc200357873)

[1.1 Анализ требований 4](#_Toc200357874)

[1.2 Обзор технологий для разработки графического интерфейса 5](#_Toc200357875)

[1.3 Вывод по первому разделу 6](#_Toc200357876)

[2 Проектирование приложения 7](#_Toc200357877)

[2.1 Анализ сущностей и их связей 7](#_Toc200357878)

[2.2 Архитектура приложения 8](#_Toc200357879)

[2.3 Проектирование пользовательского интерфейса 9](#_Toc200357880)

[2.4 Вывод по второму разделу 12](#_Toc200357881)

[3 Создание и тестирование приложения 14](#_Toc200357882)

[3.1 Реализация графического интерфейса 16](#_Toc200357883)

[3.2 Реализация инициализации базы данных 20](#_Toc200357884)

[3.3 Запуск приложения 20](#_Toc200357885)

[3.4 Тестирование и отладка 20](#_Toc200357886)

[3.5 Вывод по третьему разделу 25](#_Toc200357887)

[Приложение А (обязательное) Листинг app.py 27](#_Toc200357888)

[Приложение Б (обязательное) Листинг database.py 28](#_Toc200357889)

[Приложение В (обязательное) Листинг models.py 30](#_Toc200357890)

[Приложение Г (обязательное) Листинг view.py 33](#_Toc200357891)

[Приложение Д (обязательное) Листинг seeder.py 38](#_Toc200357892)

[Приложение И (справочное) Библиографический список 39](#_Toc200357893)

Введение

Цель — разработка и реализация приложения, изучение технологий создания приложений с графическим интерфейсом, изучение объектно-ориентированного подхода в создании приложений. Приложение предназначено для просмотра, редактирования, книг и аудиокниг, вычисления средней и медианной стоимости книг и аудиокниг.

Задачи практики:

* изучение современных технологий для разработки приложений с графическим интерфейсом;
* разработка интерфейса с учётом принципов удобного пользовательского взаимодействия;
* реализация необходимой функциональности.

# Анализ проблемы и требований, обзор технологий

В данной главе проводится анализ проблемы и требований к графическим приложениям, а также рассматриваются технологии, подходящие для их реализации. Обосновывается необходимость разработки приложения и определяются основные требования к его функциональности. Также проводится обзор современных технологий для создания графических интерфейсов, их преимуществ и возможных ограничений.

## Анализ требований

Требования к приложению вытекают из задач информационной системы для хранения данных. Система должна обеспечивать хранение данных о книгах и аудиокнигах, а также предоставлять удобный интерфейс для их просмотра, добавления, удаления, редактирования.

Функциональные требования:

* возможность просмотра информации о книгах (наименование, стоимость, автор);
* возможность просмотра информации об аудиокнигах (наименование, стоимость, автор, диктор, длительность);
* функция редактирования данных с последующим сохранением изменений;
* возможность расчета средней и медианной стоимости.

Требования к интерфейсу:

* простота и удобство — интерфейс должен быть понятным для пользователя;
* визуальное разделение информации — данные должны быть структурированы, важные элементы выделены;
* интерактивность — система должна оперативно реагировать на изменения (например, обновление данных после редактирования).

Для реализации приложения, соответствующего сформулированным требованиям, необходимо выбрать подходящие технологии.

## Обзор технологий для разработки графического интерфейса

Для создания приложения рассматривались три технологии: **C# с Avalonia**, **Java с Swing** и **Python с Tkinter**. Каждая из них обладает своими особенностями, преимуществами и ограничениями, которые влияют на возможность их использования в данном проекте.

C# вместе с фреймворком Avalonia предоставляет кроссплатформенное решение для создания настольных приложений. Avalonia использует XAML для описания интерфейса, что упрощает разработку сложных UI.

Плюсы:

* высокая производительность;
* хорошая интеграция с .NET;
* поддержка MVVM.

Минусы:

* сложность обучения;
* низкая популярность;
* поддержка linux ограничена.

Java и Swing - проверенная временем комбинация для создания десктопных приложений.

Плюсы:

* кроссплатформенность;
* большое количество учебных материалов.

Минусы:

* устаревший внешний вид;
* сложность в создании современного UI;
* многословный код.

Python в связке с Tkinter - оптимальный выбор для быстрой разработки простых настольных приложений.

Плюсы:

* низкий порог входа;
* минимальная настройка окружения;
* Tkinter является встроенной библиотекой в стандартной поставке Python;
* простота кода, хорошая документация.

Минусы:

* ограниченные возможности кастомизации интерфейса;
* средняя производительность при работе с большими данными.

## Вывод по первому разделу

Для учебного проекта по управлению коллекцией книг и аудиокниг предпочтительнее выбрать Python с Tkinter. Этот стек обеспечивает максимальную простоту разработки при достаточной функционалности, имеет низкий порог входа и не требует сложной настройки окружения. C# с Avalonia избыточен для такой задачи, а Java с Swing морально устарел. Tkinter позволяет быстро получить рабочий прототип с минимальными затратами времени на изучение технологии.

# Проектирование приложения

Проектирование приложения является ключевым этапом разработки, на котором определяются его структура и функциональность. В данном разделе рассматриваются основные аспекты проектирования, включая анализ сущностей и их связей, выбор архитектуры и проектирование интерфейсов. Результатом проектирования станет чёткое понимание того, как приложение будет функционировать.

## Анализ сущностей и их связей

Перед тем как приступить к проектированию приложения необходимо провести анализ сущностей, которые будут использоваться в системе, и их взаимосвязей. Это позволит определить, какие данные будут обрабатываться приложением.

Прежде нужно определить основные сущности, такие как «Книга» и «Аудиокнига». На рисунке 1 представлена диаграмма сущностей.

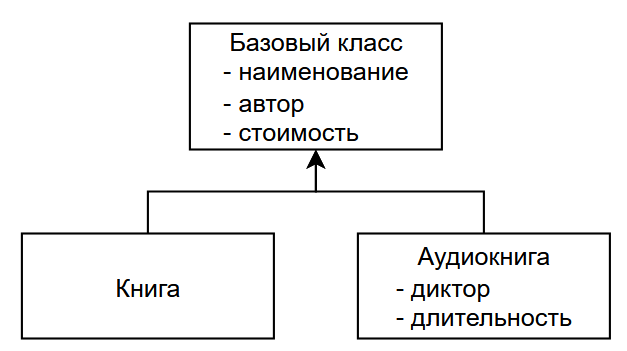


Рисунок 1 – Диаграмма сущностей приложения

Сущности «Книга» и «Аудиокнига» являются наследниками базового класса. Это предоставляет возможность хранить сущности в гомогенных контейнерах и обрабатывать их как сущности базового класса. Такая возможность нам понадобится при извлечении сущностей из базы данных и отображение их в пользовательском интерфейсе в общем списке.

## Архитектура приложения

Приложение разделено на независимые друг от друга компоненты, которые могут быть использованы повторно. Это упрощает разработку и поддержку кода.

Суть в том, что приложение разделёно на логические слои, каждый из которых отвечает за определённую функциональность. Представление – графический интерфейс. Модели – сущности приложения «Книга» и «Аудиокнига» и слой базы данных. Схема архитектуры приложения представлена на рисунке 2.

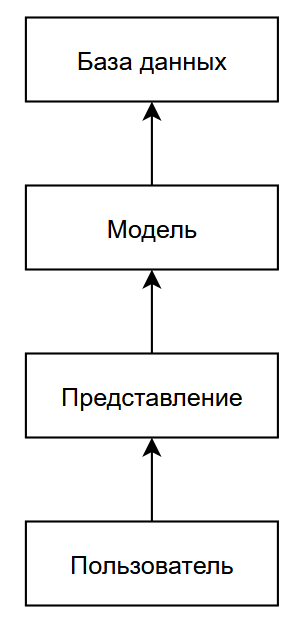


Рисунок 2 – Архитектура приложения

## Проектирование пользовательского интерфейса

При разработке интерфейса приложения был использован минималистичный подход с акцентом на функциональность. Основной экран содержит список книг и аудиокниг в виде таблицы, где четко отображаются ключевые атрибуты каждой сущности. Все операции (добавление, редактирование, расчет статистики) вынесены в модальные окна, что позволяет сохранить чистоту основного интерфейса и избежать перегруженности.

### Основные элементы интерфейса

Приложение должно включать следующие элементы:

* список сущностей;
* панель управления для взаимодействия с сущностями;
* форма добавления книги;
* форма добавления аудиокниги;
* форма редактирования;
* модальное окно расчета средней и медианной стоимости книг;
* модальные окна для вывода ошибок при попытке выполнить действия над сущностями (редактирование, расчет средней и медианной стоимости), окна выводятся, когда пользователь не выбрал не одной сущности.

Список сущностей должен иметь вид таблицы и содержать колонки:

* + id сущности;
  + наименование;
  + автор;
  + стоимость;
  + диктор (пустое для сущности «Книга»);
  + длительность (пустое для сущности «Книга»).

Панель управления для взаимодействия сущностями состоит из кнопок:

* + кнопка добавления книги;
  + кнопка добавления аудиокниги;
  + кнопка редактирования;
  + кнопка удаления;
  + кнопка расчета средней и медианной стоимости.

Форма добавления книги, состоит из трех модальных окон вызываемых друг за другом:

* + окно ввода названия книги;
  + окно ввода автора книги;
  + окно ввода стоимости книги.

Форма добавления аудиокниги, состоит из таких же модальных окон что и форма добавления книги, плюс:

* + окно ввода диктора;
  + окно ввода длительности.

Форма редактирования, аналогична формам добавления сущностей, в каждом окне подставлены данные текущей сущности;

### Принципы UX/UI

Интерфейс должен быть интуитивно понятным даже для пользователей, не обладающих техническими знаниями; все элементы интерфейса (поля, кнопки) должны быть легко доступны и понятны. Список «Книг» и «Аудиокниг» отделен от блока управления. Интерфейс должен оперативно реагировать на действия пользователя (например, мгновенное отображение изменений после редактирования).

### Проектирование интерфейса

Определим макет, который визуализирует расположение основных элементов. Этот макет поможет определить, где будут находиться текстовые поля, кнопки и другие элементы управления, чтобы обеспечить удобство использования и логичную структуру интерфейса.

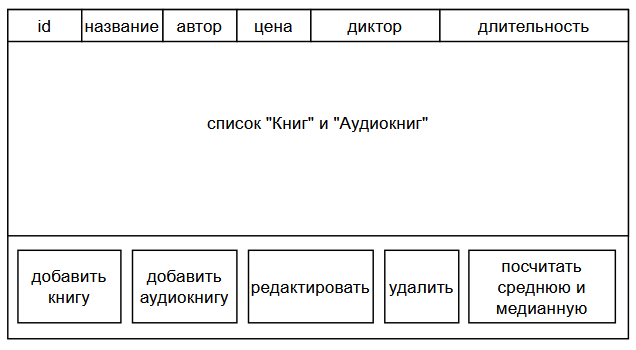


Рисунок 3 – Макет главного экрана интерфейса

Определим макет модального окна, которое используется для, добавления и редактирования сущностей.

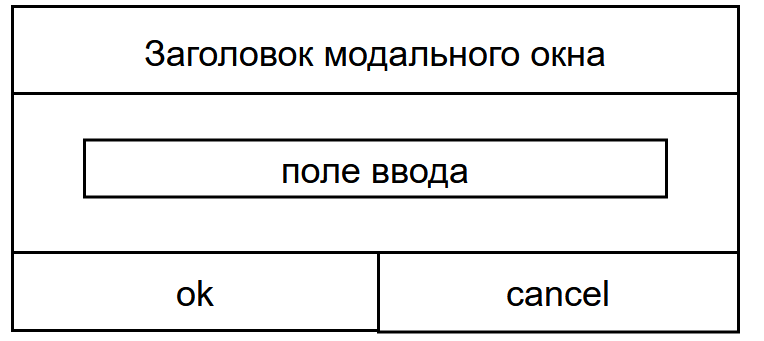


Рисунок 4 – Макет модального окна для ввода информации

Определим макет модального окна, которое используется для вывода информации.

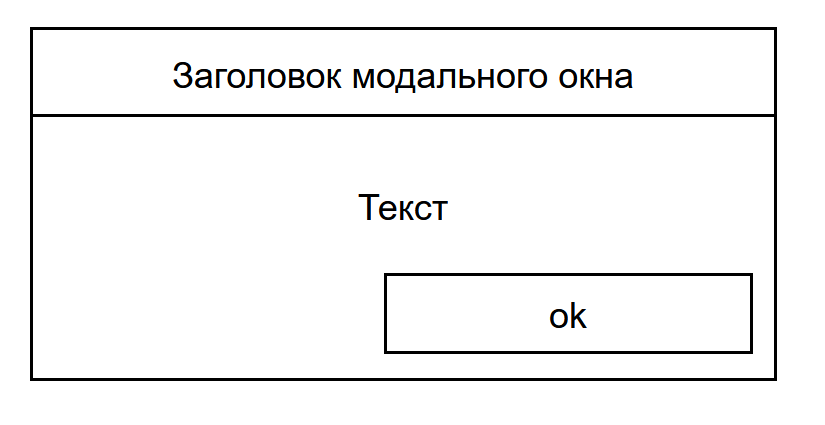


Рисунок 5 – Макет модального окна для вывода информации

В результате проектирования пользовательского интерфейса создан удобный и интуитивно понятный интерфейс, который обеспечивает эффективное взаимодействие пользователя с системой. На этапе проектирования разработана макетная схема, включающая ключевые элементы, такие как: главный экран, модальное окно ввода информации и модальное окно вывода информации.

Учёт принципов UX/UI, таких как простота, визуальное разделение информации и интерактивность, позволил сделать интерфейс максимально удобным для использования. Расположение элементов в макете логично и интуитивно понятно, что способствует улучшению пользовательского опыта и упрощает взаимодействие с компонентом.

## Вывод по второму разделу

В данном разделе cпроектировано приложение. На этапе проектирования проведён анализ сущностей и их связей, что позволило определить структуру данных и их взаимодействие в системе. Для наглядности создана диаграмма сущностей приложения, которая визуализирует основные сущности и их взаимосвязи.

В качестве основы для разработки выбран компонентный подход, который обеспечивает модульность и гибкость приложения. Определены ключевые элементы и функции приложения, такие как отображение и редактирование данных о книгах и аудиокнигах.

Кроме того, при проектировании учтены современные стандарты UI/UX, что обеспечивает удобство и интуитивность взаимодействия пользователя с интерфейсом.

# Создание и тестирование приложения

В данном разделе описывается процесс создания и тестирования приложения. На основе анализа требований и проектирования, проведённых в предыдущих разделах, была выполнена практическая реализация приложения. Результатом является готовое приложение.

Для разработки необходимо установить python 3.13. Установка базы данных sqlite3 и библиотеки графического интерфейса tkinter не требуется, потому что эти компоненты входят в стандартную библиотеку python.

Для начала определим схему базы данных. База данных состоит из одной таблицы entites и ее схема представлена на рисунке 6.

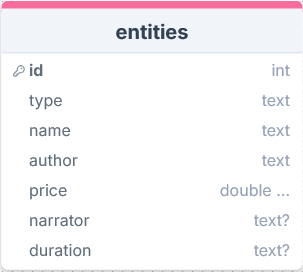


Рисунок 6 – схема базы данных

Для начала напишем класс Database, который отвечает за работу с sqlite. В нем реализованы основные CRUD-операции (создание таблицы, добавление, обновление, удаление и получение записей) через статические методы. Класс использует единое подключение к базе данных и обеспечивает хранение как книг, так и аудиокниг в одной таблице, различая их по полю type.

Затем создадим иерархия классов сущностей. Базовый абстрактный класс Entity определяет общую структуру и методы работы с данными, а классы-потомки Book и AudioBook реализуют специфичную для каждого типа логику. Поля id, name, author, price содержатся в классе Entity. Класс Book реализует метод type, который служит для определения типа и возвращает строковый идентификатор «book». Класс Audiobook так же содержит метод type, который возвращает строковый идентификатор «audiobook». Класс AudioBook дополнительно содержит поля duration и narrator. Оба класса содержат метод from\_row который создает объект из переданных параметров. Все классы связаны с Database, но делегируют ему операции с хранилищем.

Особенностью реализации стала система регистрации типов через \_\_init\_subclass\_\_, позволяющая автоматически определять класс для создания объекта при загрузке данных из базы. Когда интерпретатор встречает объявление класса Book(Entity) или AudioBook(Entity), перед созданием класса вызывается Entity.\_\_init\_subclass\_\_, в этом методе формируется словарь Entity.registry, который в итоге будет иметь вид {'book': Book, 'audiobook': AudioBook}. Это позволяет позже находить нужный класс по его типу при загрузке данных из БД, не используя проверку типов и соблюдая принцип открытости-закрытости. Диаграмма сущностей и их отношений представлена на рисунке 7.

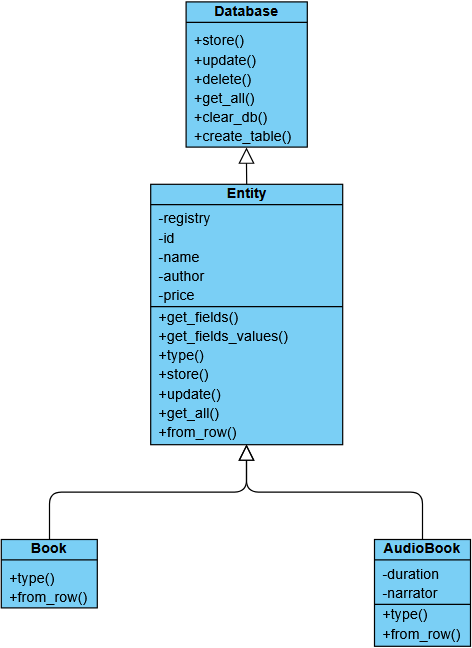


Рисунок 7 – Диаграмма отношений сущностей

## Реализация графического интерфейса

После реализации базовых сущностей, начинается разработка интерфейса приложения. Графический интерфейс выполнен с помощью библиотеки tkinter.

### Реализация представления

При разработке класса графического интерфейса был создан класс Tinker реализующий интерфейс View. В методе \_\_init\_\_ класса Tinker сначала создается основное окно приложения с помощью tk.Tk(). Оно получает заголовок "Библиотека", после чего последовательно вызываются методы list\_widget и buttons\_widget и refresh.

Метод list\_widget создаёт таблицу с помощью виджета ttk.Treeview. Эта таблица предназначена для отображения информации о книгах и аудиокнигах. В таблице определяются следующие колонки:

* type это скрытая колонка, содержащая технический тип сущности («book» или «audiobook»);
* id уникальный идентификатор записи;
* Наименование;
* Автор;
* Стоимость;
* Диктор;
* Длительность.

Для каждой колонки задаются заголовки и ширина, после чего таблица размещается в окне с помощью метода pack библиотеки Tkinter с параметрами fill=tk.BOTH, expand=True, что позволяет ей масштабироваться вместе с окном. Дополнительно колонка type устанавливается как невидимая (width=0, stretch=False), чтобы скрыть служебную информацию от пользователя, но оставить доступной для логики обработки.

Метод buttons\_widget создаёт фрейм, в котором размещаются кнопки управления:

* **Добавить книгу;**
* **Добавить аудиокнигу;**
* **Редактировать;**
* **Удалить;**
* **Посчитать среднюю и медианную стоимость.**

Каждая кнопка создаётся с помощью виджета ttk.Button из модуля ttk библиотеки Tkinter и привязывается к своему обработчику с помощью параметра command, ссылающегося на метод соответствующего действия (add\_book, add\_audiobook, edit\_selected, delete\_selected, calculate\_stats). Кнопки размещаются в одну строку с отступами между ними с помощью сетки grid, метод grid является встроенным методом библиотеки Tkinter для размещения виджетов в табличной (сеточной) структуре, позволяет указывать координаты ячеек через параметры row и column. В данном случае каждая кнопка размещается в одной строке (row=0), но в своей отдельной колонке (column=0, column=1 и т. д.), что обеспечивает ровную горизонтальную раскладку с равномерными промежутками между кнопками.

Метод refresh отвечает за синхронизацию таблицы с текущими данными. При каждом вызове он очищает все строки в Treeview, после чего запрашивает актуальный список сущностей с помощью метода Entity.get\_all. Каждый объект преобразуется в список значений с помощью метода entity.get\_fields\_value, и в таблицу добавляется новая строка.

Затем реализованы основные функции управления сущностями через модальные окна. При добавлении или редактировании книги или аудиокниги, интерфейс вызывает диалоговые окна simpledialog.askstring и askfloat для получения данных от пользователя. В зависимости сущности от типа создаётся экземпляр соответствующего класса (Book или AudioBook), который затем сохраняется в базу с помощью методов store или update. После каждого такого действия вызывается refresh для обновления отображения.

Удаление реализовано через метод delete\_selected, который получает идентификаторы выделенных строк, вызывает метод Entity.delete для каждой из них, а затем также обновляет интерфейс через метод refresh.

Вся логика обновления, добавления и удаления основана на выборе пользователем одной или нескольких строк в Treeview. Работа с выбором осуществляется через self.tree.selection.

Метод calculate\_stats реализует дополнительную функцию анализа данных — вычисление средней стоимости и медианы среди выделенных пользователем элементов таблицы. Сначала определяется список выделенных строк, из каждой из которых извлекается значение из пятой колонки таблицы Treeview, которая соответствует стоимости книги или аудиокниги. Значения приводятся к числовому типу и добавляются в список цен. Далее, если список не пуст, вычисляется средняя стоимость как сумма всех значений, делённая на их количество, а медиана определяется с использованием встроенной функции statistics.median из стандартной библиотеки Python statistics, которая возвращает срединное значение в отсортированном списке. Результаты округляются до двух знаков после запятой и выводятся пользователю через информационное окно, созданное с помощью метода messagebox.showinfo, которое отображает результаты в отдельном всплывающем окне с заголовком и текстом.

Диаграмма классов графического интерфейса представлена на рисунке 8.

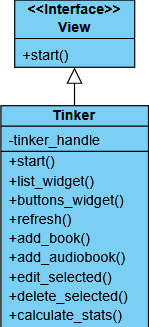


Рисунок 8 – Диаграмма классов графического интерфейса

## Реализация инициализации базы данных

Инициализация базы данных сделана через служебный скрипт. В нем реализована функцию seed\_data(), которая создает несколько популярных книг и аудиокниг с разными параметрами (название, автор, цена), сразу сохраняя их в базу через метод store(). Особенность - поддержка флага --clear при запуске, который полностью очищает базу перед заполнением, что удобно для тестирования. Скрипт автоматически создает таблицу при первом запуске и выводит статус выполнения в консоль.

## Запуск приложения

Точка входа в приложения реализована через создание экземпляра класса Tinker (который содержит весь графический интерфейс) и запускает главный цикл обработки событий через метод start()

## Тестирование и отладка

После завершения разработки приложения необходимо провести тестирование, чтобы убедиться в его корректной работе. Процесс тестирования включает проверку отображения данных, функциональности форм редактирования и сохранения изменений.

Для начала тестирования необходимо запустить инициализацию тестовых данных используя команду: «python seeder.py**»**

При успешном выполнение, скрипт выдаст строку «База наполнена», пример запуска на рисунке 9.



Рисунок 9 – Инициализация тестовых данных

Если нужно очистить базу данных, необходимо вызвать эту же команду с флагом –clear: «python seeder.py –clear**»**

При успешном выполнении скрипт выдаст две строки «База очищена» и «База наполнена», пример запуска на рисунке 10.

****

Рисунок 10 – Инициализация тестовых данных с предварительной очисткой базы

Для запуска приложения нужно выполнить команду: «python app.py**».** После чего будет выведен стартовый экран приложения, рисунок 11.

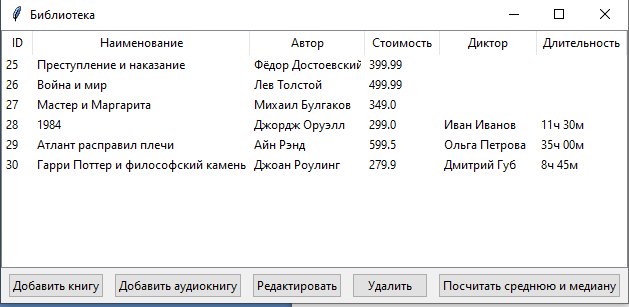
****

Рисунок 11 – Стартовый экран приложения

При добавлении книги, пользователю будет последовательно выведены 3 модальных окна, после их заполнения, новая книга добавится в общий список, рисунки 12 и 13.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| а | б | в |
|  |  |  |

Рисунок 12 – Модальные окна добавления книги

После заполнения данных, книга появится в списке, рисунок 13.

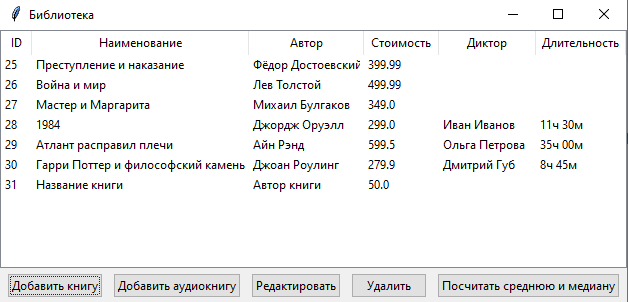


Рисунок 13 – Результат добавления книги

При добавлении аудиокниги, пользователю будет последовательно выведено 5 модальных окон, после чего аудиокнига будет добавлена в общий список, рисунки 14 и 15.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| а | б | в | г | д |

Рисунок 14 – Модальные окна добавления аудиокниги

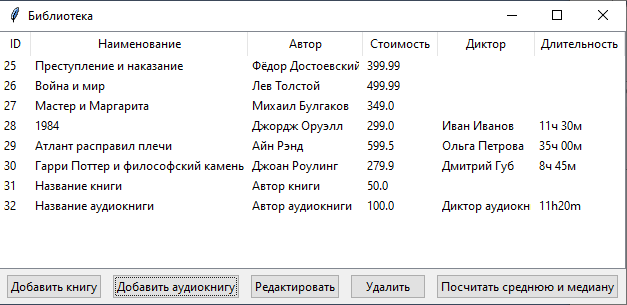


Рисунок 15 – Добавление аудиокниги

При редактировании сущности пользователю будут выведены модальны окна с полями характерными для выбранной сущности, с представленными полями, после чего сущность будет обновлена в списке, рисунки 16 и 17.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| а | б | в |

Рисунок 16 – Модальные окна Редактирование сущностей

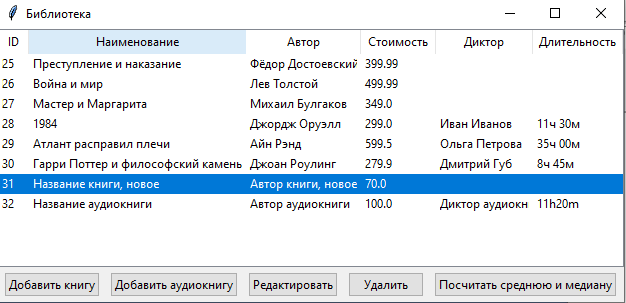


Рисунок 17 – Результат редактирование сущностей

Для удаления сущности, особых модальных окон не предсмотренно, сущность просто пропадает из списка.

Вычисление средней арифметической и медианной стоимости для выбранных элементов списка показано на рисунках 18 и 19.

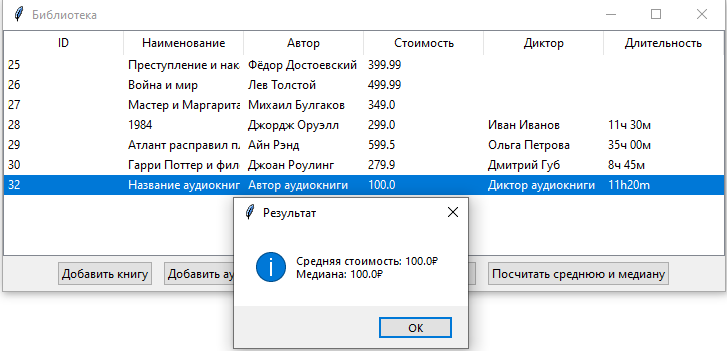


Рисунок 18 – Вычисление средней арифметический и медианной для одного элемента

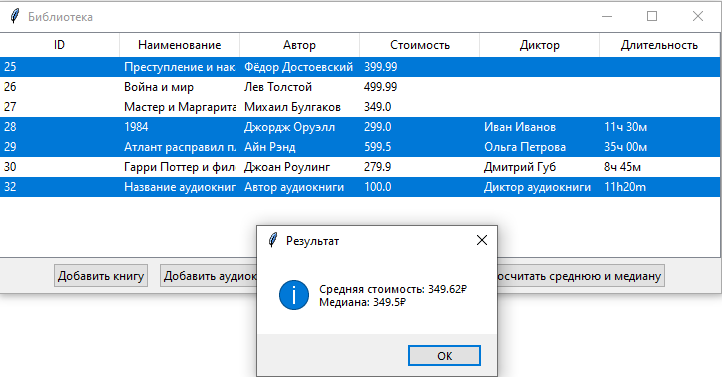


Рисунок 19 – Вычисление средней арифметический и медианной для нескольких элементов

Модальные окна вывода ошибок и информации представлены на рисунках 20 и 21.

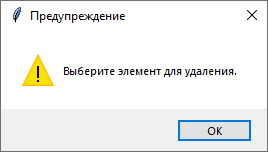


Рисунок 20 – Модальное окно с ошибкой

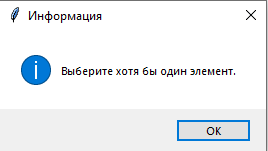


Рисунок 21 – Модальное оккно с информацией

## Вывод по третьему разделу

В данном разделе была проведена практическая реализация приложения. Были разработаны диаграмма классов, которые помогли структурировать данные и определить связи между сущностями.

В процессе разработки создан и реализован графический интерфейс, обеспечивающий удобное отображение и редактирование информации. Проведено тестирование функциональности, подтвердившее корректность работы приложения. Итогом работы стало готовое приложение, соответствующие требованиям проекта.

Заключение

В ходе выполнения данной практической работы была проведена разработка приложения с графическим интерфейсом.

На первом этапе был выполнен анализ технологий для реализации приложений с графическим интерфейсом. На основе анализа были сформированы ключевые требования к приложению.

На втором этапе было выполнено проектирование приложения, которое помогло определиться необходимые элементы для функционирования приложения.

Далее была проведена разработка приложения, включающая создание проекта, и реализацию логики. В ходе тестирования были проверены сценарии корректного отображения, редактирования и сохранения и расчета данных.

Итогом работы стало создание полноценного, приложения с графическим интерфейсом.

1. (обязательное)  
   Листинг app.py

from view import Tinker

if \_\_name\_\_ == "\_\_main\_\_":

tinker = Tinker()

tinker.start()

1. (обязательное)  
   Листинг database.py

import sqlite3

class Database:

conn = sqlite3.connect('library.db')

conn.row\_factory = sqlite3.Row

@staticmethod

def create\_table():

Database.conn.execute('''

CREATE TABLE IF NOT EXISTS entities (

id INTEGER PRIMARY KEY AUTOINCREMENT,

type TEXT NOT NULL,

name TEXT NOT NULL,

author TEXT NOT NULL,

price REAL NOT NULL,

narrator TEXT,

duration TEXT

)

''')

Database.conn.commit()

@staticmethod

def store(entity):

fields = entity.get\_fields()

fields.pop('id', None)

fields['type'] = entity.type()

column\_names = ", ".join(fields.keys())

placeholders = ", ".join("?" for \_ in fields)

Database.conn.execute(

f'INSERT INTO entities ({column\_names}) VALUES ({placeholders})',

tuple(fields.values())

)

Database.conn.commit()

@staticmethod

def update(entity):

fields = entity.get\_fields()

fields.pop('id', None)

set\_clause = ", ".join(f"{field}=?" for field in fields.keys())

sql = f"UPDATE entities SET {set\_clause} WHERE id=?"

Database.conn.execute(sql, list(fields.values()) + [entity.id])

Database.conn.commit()

@staticmethod

def delete(entity\_id):

Database.conn.execute('DELETE FROM entities WHERE id=?', (entity\_id,))

Database.conn.commit()

@staticmethod

def get\_all():

cursor = Database.conn.execute('SELECT \* FROM entities')

return cursor.fetchall()

@staticmethod

def clear\_db():

Database.conn.execute('DELETE FROM entities')

Database.conn.commit()

1. (обязательное)  
   Листинг models.py

from database import Database

from abc import ABC, abstractmethod

class Entity(Database, ABC):

registry = {}

def \_\_init\_subclass\_\_(cls, \*\*kwargs):

super().\_\_init\_subclass\_\_(\*\*kwargs)

if hasattr(cls, 'type') and callable(cls.type):

Entity.registry[cls.type()] = cls

def \_\_init\_\_(self, entity\_id, name, author, price):

self.id = entity\_id

self.name = name

self.author = author

self.price = price

@classmethod

@abstractmethod

def from\_row(cls, row):

pass

def get\_fields(self):

return vars(self).copy()

def get\_fields\_values(self):

return list(self.get\_fields().values())

@staticmethod

@abstractmethod

def type():

pass

def store(self):

Database.store(self)

def update(self):

Database.update(self)

@classmethod

def get\_all(cls):

rows = Database.get\_all()

items = []

for row in rows:

type\_ = row["type"]

klass = cls.registry.get(type\_)

if klass:

items.append(klass.from\_row(row))

return items

class Book(Entity):

def \_\_init\_\_(self, entity\_id, name, author, price):

super().\_\_init\_\_(entity\_id, name, author, price)

@staticmethod

def type():

return "book"

@classmethod

def from\_row(cls, row):

id\_, \_, name, author, price, \_, \_ = row

return cls(id\_, name, author, price)

class AudioBook(Entity):

def \_\_init\_\_(self, entity\_id, name, author, price, narrator, duration):

super().\_\_init\_\_(entity\_id, name, author, price)

self.narrator = narrator

self.duration = duration

@staticmethod

def type():

return "audiobook"

@classmethod

def from\_row(cls, row):

id\_, \_, name, author, price, narrator, duration = row

return cls(id\_, name, author, price, narrator, duration)

1. (обязательное)  
   Листинг view.py

from abc import ABC, abstractmethod

import tkinter as tk

from tkinter import ttk, messagebox, simpledialog

import statistics

from models import Entity, Book, AudioBook

class View(ABC):

@abstractmethod

def start(self):

pass

class Tinker(View):

tinker\_handle = None

def \_\_init\_\_(self):

self.tinker\_handle = tk.Tk()

self.tinker\_handle.title("Библиотека")

self.list\_widget()

self.buttons\_widget()

self.refresh()

def start(self):

self.tinker\_handle.mainloop()

def list\_widget(self):

columns = ('type', 'ID', 'Наименование', 'Автор', 'Стоимость', 'Диктор', 'Длительность')

self.tree = ttk.Treeview(self.tinker\_handle, columns=columns, show='headings', selectmode='extended')

for col in columns:

self.tree.heading(col, text=col)

self.tree.column(col, width=120)

self.tree.pack(fill=tk.BOTH, expand=True)

self.tree.column("type", width=0, stretch=False)

def buttons\_widget(self):

btn\_frame = tk.Frame(self.tinker\_handle)

btn\_frame.pack(pady=5)

ttk.Button(btn\_frame, text="Добавить книгу", command=self.add\_book).grid(row=0, column=0, padx=5)

ttk.Button(btn\_frame, text="Добавить аудиокнигу", command=self.add\_audiobook).grid(row=0, column=1, padx=5)

ttk.Button(btn\_frame, text="Редактировать", command=self.edit\_selected).grid(row=0, column=2, padx=5)

ttk.Button(btn\_frame, text="Удалить", command=self.delete\_selected).grid(row=0, column=3, padx=5)

ttk.Button(btn\_frame, text="Посчитать среднюю и медиану", command=self.calculate\_stats).grid(row=0, column=4, padx=5)

def refresh(self):

for i in self.tree.get\_children():

self.tree.delete(i)

for entity in Entity.get\_all():

values = [entity.type()] + entity.get\_fields\_values()

self.tree.insert('', tk.END, values=values)

def add\_book(self):

name = simpledialog.askstring("Добавить книгу", "Наименование:")

author = simpledialog.askstring("Автор", "Автор:")

price = simpledialog.askfloat("Стоимость", "Стоимость:")

if name and author and price is not None:

Book(None, name, author, price).store()

self.refresh()

def add\_audiobook(self):

name = simpledialog.askstring("Добавить аудиокнигу", "Наименование:")

author = simpledialog.askstring("Автор", "Автор:")

price = simpledialog.askfloat("Стоимость", "Стоимость:")

narrator = simpledialog.askstring("Диктор", "Диктор:")

duration = simpledialog.askstring("Длительность", "Длительность:")

if name and author and price is not None and narrator and duration:

AudioBook(None, name, author, price, narrator, duration).store()

self.refresh()

def edit\_selected(self):

selected = self.tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите элемент для редактирования.")

return

item = self.tree.item(selected[0])['values']

type\_ = item[0]

entity\_id = item[1]

name = simpledialog.askstring("Редактировать", "Наименование:", initialvalue=item[2])

author = simpledialog.askstring("Автор", "Автор:", initialvalue=item[3])

price = simpledialog.askfloat("Стоимость", "Стоимость:", initialvalue=item[4])

if type\_ == Book.type():

if name and author and price is not None:

Book(entity\_id, name, author, price).update()

else:

narrator = simpledialog.askstring("Диктор", "Диктор:", initialvalue=item[5])

duration = simpledialog.askstring("Длительность", "Длительность:", initialvalue=item[6])

if name and author and price is not None and narrator and duration:

AudioBook(entity\_id, name, author, price, narrator, duration).update()

self.refresh()

def delete\_selected(self):

selected = self.tree.selection()

if not selected:

messagebox.showwarning("Предупреждение", "Выберите элемент для удаления.")

return

for sel in selected:

item = self.tree.item(sel)

entity\_id = item['values'][1]

Entity.delete(entity\_id)

self.refresh()

def calculate\_stats(self):

selected\_items = self.tree.selection()

if not selected\_items:

messagebox.showinfo("Информация", "Выберите хотя бы один элемент.")

return

prices = []

for item\_id in selected\_items:

item = self.tree.item(item\_id)

values = item['values']

try:

price = float(values[4])

prices.append(price)

except (IndexError, ValueError):

continue

if not prices:

messagebox.showwarning("Ошибка", "Не удалось получить стоимость из выбранных элементов.")

return

avg = round(sum(prices) / len(prices), 2)

med = round(statistics.median(prices), 2)

messagebox.showinfo("Результат", f"Средняя стоимость: {avg}₽\nМедиана: {med}₽")

1. (обязательное)  
   Листинг seeder.py

import sys

from database import Database

from models import Book, AudioBook

def seed\_data():

Book(None, 'Преступление и наказание', 'Фёдор Достоевский', 399.99).store()

Book(None, 'Война и мир', 'Лев Толстой', 499.99).store()

Book(None, 'Мастер и Маргарита', 'Михаил Булгаков', 349.00).store()

AudioBook(None, '1984', 'Джордж Оруэлл', 299.00, 'Иван Иванов', '11ч 30м').store()

AudioBook(None, 'Атлант расправил плечи', 'Айн Рэнд', 599.50, 'Ольга Петрова', '35ч 00м').store()

AudioBook(None, 'Гарри Поттер и философский камень', 'Джоан Роулинг', 279.90, 'Дмитрий Губ', '8ч 45м').store()

if \_\_name\_\_ == "\_\_main\_\_":

clear\_flag = '--clear' in sys.argv

if clear\_flag:

Database.clear\_db()

print("База очищена")

Database.create\_table()

seed\_data()

print("База наполнена")

1. (справочное)  
   Библиографический список
2. Официальная документация по Python [Электронный ресурс]. – Дата обращения: 02.06.2025. – Доступ из: <https://docs.python.org/3/>.
3. Официальная документация по Tkinker [Электронный ресурс]. – Дата обращения: 02.06.2025. – Доступ из: <https://tkdocs.com/>.
4. Официальная документация по Sqlite [Электронный ресурс]. – Дата обращения: 02.06.2025. – Доступ из: <https://sqlite.org/docs.html>.
5. Ресурс по проектированию схем для дизайна интерфейса [Электронный ресурс]. – Дата обращения: 05.06.2025. – Доступ из: <https://app.diagrams.net>.
6. Ресурс по проектированию базы данных [Электронный ресурс]. – Дата обращения: 05.06.2025. – Доступ из: <https://drawsql.app>.
7. Ресурс по проектированию UML диаграмм [Электронный ресурс]. – Дата обращения: 05.06.2025. – Доступ из: <https://online.visual-paradigm.com>