Пояснительная записка

К проекту на тему:

«Разработка приложения для библиотеки»

Выполнил: Попов Александр

Руководитель: Осипов Антон

г. Сургут 2020г.

Содержание:

1.Введение

На уроках в старшей школе постоянно требуется наличие тех или иных справочных материалов, способствующих обучающему процессу. Эти справочные материалы, зачастую, приходится брать в школьной библиотеке, которая не всегда справляется с нагрузкой. Из-за этого приходится ждать довольно большое количество времени, затрачиваемое библиотекарем на поиск необходимого билета и занесение всей информации. Я решил разработать приложение, которое упростит и сократит процесс выдачи книги.

Проблема проекта: избыточная длительность и сложность процесса регистрации выдачи книг в библиотеках.

Цель проекта: разработать приложение на языке программирования «Python», упрощающее процесс регистрации выдачи книг в библиотеках.

Задачи:

- Изучить возможные архитектурные паттерны проектирования
- Изучить модули, необходимые для разработки приложения
- Разработать рабочую версию приложения

2.Теоретическая часть

2.1. Историческая справка

Библиотеки впервые появились на древнем Востоке. Обычно первой библиотекой называют собрание глиняных табличек, приблизительно 2500 год до н. э., найденное в храме шумерского города Ниппур. В одной из гробниц близ египетских Фив был обнаружен ящик с папирусами времени II переходного периода (XVIII—XVII вв. до н. э.). В эпоху Нового царства Рамсесом II было собрано около 20 000 папирусов. Самая известная древневосточная библиотека — собрание клинописных табличек из дворца ассирийского царя VII века до н. э. Ашшурбанипала в Ниневии. Основная часть табличек содержит юридическую информацию. В древней Греции первая публичная библиотека была основана в Гераклее тираном Клеархом (IV век до н. э.).

Крупнейшим центром античной книжности стала Александрийская библиотека. Она была создана в III веке до н. э. Птолемеем I и была центром образования всего эллинистического мира. Александрийская библиотека являлась частью комплекса mouseĩon (музей). В комплекс входили жилые комнаты, столовые помещения, помещения для чтения, ботанический и зоологический сады, обсерватория и библиотека. Позднее к нему были добавлены медицинские и астрономические инструменты, чучела животных, статуи и бюсты, которые были использованы для обучения. В mouseĩon входило 200 000 папирусов в Храме (почти все библиотеки античности были при храмах) и 700 000 документов в Школе. Музей и большая часть Александрийской библиотеки были уничтожены приблизительно в 270 году нашей эры.

В Средние века очагами книжности были монастырские библиотеки, при которых действовали скриптории (смотри, в частности, статью Библиотеки эпохи Каролингов). Там переписывалось не только Священное писание и сочинения Отцов Церкви, но и произведения античных авторов. В эпоху Ренессанса деятели Возрождения буквально охотились за сохранявшимися в монастырях греческими и латинскими текстами. Из-за огромной стоимости манускриптов и трудоёмкости их изготовления книги приковывались к библиотечным полкам цепями (см. книги на цепях).

Изобретение печатного станка и развитие книгопечатания внесли огромные изменения в облик и деятельность библиотек, всё более теперь отличавшихся от архивов. Библиотечные фонды начинают стремительно разрастаться. С распространением грамотности в Новое время растёт также число посетителей библиотек.

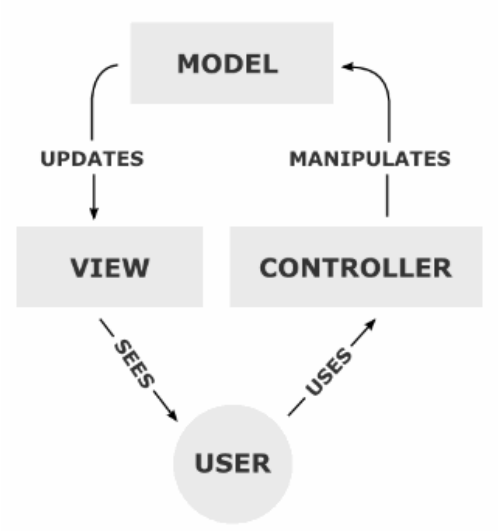
2.2. Архитектурные паттерны проектирования

Используя справочную литературу и информацию сети интернет, мне удалось найти и выделать следующие паттерны:

Model-View-Controller

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

- **Модель (Model)** предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- **Представление (View)** отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- **Контроллер (Controller)** интерпретирует действия пользователя, оповещая модель о необходимости изменений.



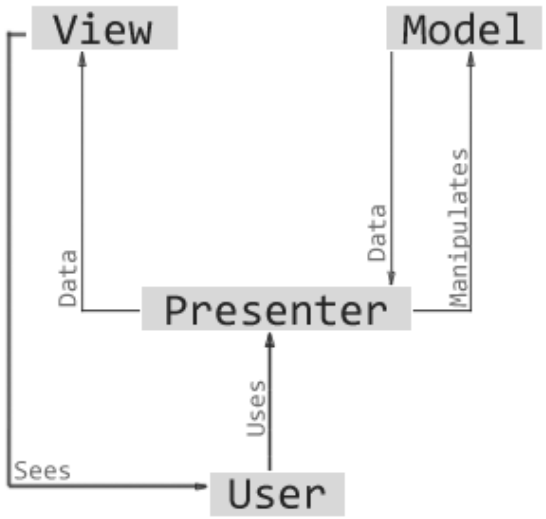
Визуализация MVC

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.

Model-View-Presenter

Model-View-Presenter (MVP) — шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения):

- **Модель (Model)** — данные для отображения;
- **Вид (View)** — реализует отображение данных (из Модели), обращается к Presenter за обновлениями, перенаправляет события от пользователя в Presenter;
- **Представитель (Presenter)** — реализует взаимодействие между Моделью и Видом (выступает в роли посредника, аналогично контроллеру в MVC) и содержит в себе всю логику представления данных о предметной области; при необходимости получает данные из хранилища и преобразует для отображения во View.



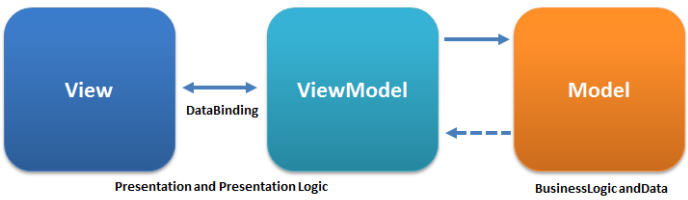
Визуализация MVP

Обычно экземпляр Вида (Представление) создаёт экземпляр Представителя, передавая ему ссылку на себя. При этом Представитель работает с Видом в абстрактном виде, через его интерфейс. Когда вызывается событие Представления, оно вызывает конкретный метод Представителя, не имеющего ни параметров, ни возвращаемого значения. Представитель получает необходимые для работы метода данные о состоянии пользовательского интерфейса через интерфейс Вида и через него же передаёт в Вид данные из Модели и другие результаты своей работы.

Model-View-ViewModel

Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model. Шаблон MVVM делится на три части:

- **Модель (Model)** (так же, как в классической MVC) представляет собой логику работы с данными и описание фундаментальных данных, необходимых для работы приложения.
- **Представление (View)** — графический интерфейс (окна, списки, кнопки и т. п.). Выступает подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью Представления. В случае, если в Модели Представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновлённое значение свойства из Модели Представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью Представления.
- **Модель Представления (ViewModel)** — с одной стороны, абстракция Представления, а с другой — обёртка данных из Модели, подлежащих связыванию. То есть, она содержит Модель, преобразованную к Представлению, а также команды, которыми может пользоваться Представление, чтобы влиять на Модель.



Визуализация MVVM

Используется для разделения модели и её представления, что необходимо для их изменения отдельно друг от друга. Например, разработчик задаёт логику работы с данными, а дизайнер работает с пользовательским интерфейсом.

Naked objects

Naked objects (оголённые объекты) — архитектурный шаблон, используемый в разработке программного обеспечения в инженерии ПО, определяется с помощью трех принципов:

- Вся бизнес-логика должна быть инкапсулирована в бизнес-объект **domain objects**. Данный принцип не является уникальной особенностью naked objects: это только строгое следование обязательствам, определенным инкапсуляцией.
- Интерфейс пользователя должен быть прямым представлением объектов предметной области (domain objects), со всеми действиями пользователя, явно содержащими создание или получение объектов предметной области и/или вызовы методов этих объектов. Данный принцип также не является уникальной особенностью naked objects: это только частная интерпретация объектно-ориентированного пользовательского интерфейса object-oriented user interface (OOUI).

Подлинная идея шаблона Naked objects возникает из комбинации обеих вышеперечисленных идей в форме третьего принципа:

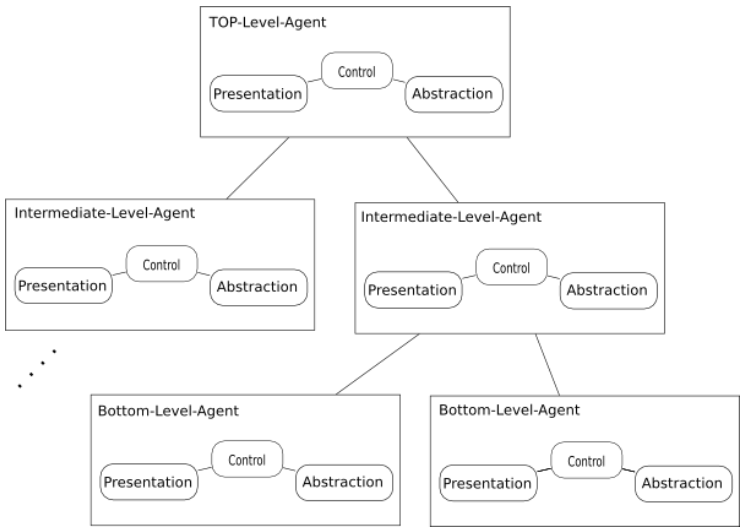
- Пользовательский интерфейс может быть сформирован полностью автоматически из определения объектов предметной области (domain objects). Данный припцип может быть реализован путём использования нескольких технологий таких, как кодогенерация и рефлексия.

Паттерн Naked objects был впервые формально определен в диссертации PhD, которая включала тщательное исследование различных предшественников шаблона, включая, например, Morphic пользовательский интерфейс.

Naked Objects обыкновенно противопоставляют шаблону **model-view-controller**. Тем не менее, опубликованная версия диссертации Поусона (Pawson) содержит предисловие Trygve Reenskaug, изобретателя шаблона model-view-controller, говорящее о том, что naked objects ближе к оригинальной идее Модель-Представление-Контроллер, чем последующие интерпретации и реализации.

Hierarchical model–view–controller

Hierarchical model–view–controller (HMVC) — Иерархические Модель-Вид-Контроллер, одно из расширений архитектурного паттерна MVC, позволяющее решить некоторые проблемы масштабируемости приложений, имеющих классическую MVC-архитектуру



Визуализация HMVC

Согласно парадигме HMVC, каждая отдельная MVC триада используется в качестве слоя в иерархической структуре. При этом каждая триада в этой иерархии независима от других, и может обратиться к контроллеру другой триады. Такой подход существенно облегчает и ускоряет разработку сложных приложений, облегчает их дальнейшую поддержку и масштабирование, способствует повторному использованию кода.

3. Практическая часть

3.1. Требования к приложению

Для решения поставленной проблемы приложение должно уметь выполнять следующие действия:

1. Хранить, добавлять, редактировать, удалять информацию о книгах
2. Хранить, добавлять, редактировать, удалять информацию об авторах книг
3. Хранить, добавлять, редактировать, удалять информацию о жанрах книг
4. Хранить, добавлять, редактировать, удалять информацию о пользователях
5. Изменять и хранить статус книг

Также приложение должно удовлетворять следующим условиям:

1. Обладать простым для понимания интерфейсом
2. Работать без ошибок и багов
3. Надежно хранить информацию

Сформировав список необходимых требований я приступил к первому этапу разработки приложения, а именно выбору необходимых бибилиотек.

3.2. Выбор библиотек для разработки приложения

Для реализации графического интерфейса было решено использовать библиотеку PyQt, потому что:

1. Разработку и редактирование окон можно производить в графическом редакторе (QtDesigner), что значительно сокращает время на разработку окон.
2. Удобное взаимодействие с объектами
3. Широкие возможности в графическом оформлении окон и их анимации.

Для хранения списков книг и пользователей, а также статус каждой книги (выдана, если да, то кому) было решено использовать СУБД (Систему Управления Базами Данных) SQlite и библиотеку её подключения – sqlite3.

Преимущества данного выбора:

1. движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы
2. Удобный язык формирования запросов

3.3. Архитектура программы.

Все графическое отображение программы будет состоять из «.uic» файлов, образующих модуль «View» программы. База данных будет полностью обособлена от интерфейса, образуя модуль «Model». Взаимодействие между этими модулями будет происходить посредствам модуля «Presenter» программы, в котором будет храниться вся логика. Объединив всё вышесказанное, можно сделать вывод, что программа будет строиться на архитектуре MVP.

Учитывая задачи программы, описанные в пункте 3.1, структура приложения будет следующей:

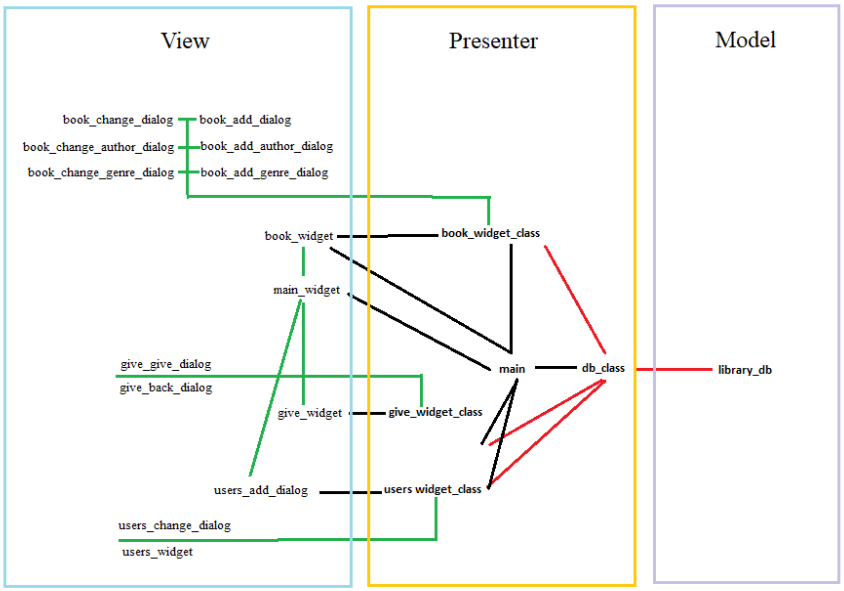
1. **Графический интерфейс (View):**
   1.1. Окно «Главное меню»
      1.1.1. Окно «Каталог»
         1.1.1.1. Вкладка «Каталог книг»
            1.1.1.1.1. Диалоговое окно «Добавление книги»
            1.1.1.1.2. Диалоговое окно «Редактирование книги»
         1.1.1.2. Вкладка «Каталог авторов»
            1.1.1.2.1. Диалоговое окно «Добавление автора»
            1.1.1.2.2. Диалоговое окно «Редактирование автора»
         1.1.1.3. Вкладка «Каталог жанров»
            1.1.1.3.1. Диалоговое окно «Добавление жанра»
            1.1.1.3.2. Диалоговое окно «Редактирование жанра»
      1.1.2. Окно «Выдача книги»
         1.1.2.1. Диалоговое окно «Выдать книгу»
         1.1.2.2. Диалоговое окно «Получить книгу»
      1.1.3. Окно «Работа с читателями»
         1.1.3.1. Диалоговое окно «Добавление читателя»
         1.1.3.2. Диалоговое окно «Редактирование читателя»
2. **Модель (Model)**
   2.1. SQlite база данных
      2.1.1. Таблица с книгами
      2.1.2. Таблица с жанрами
      2.1.3. Таблица с авторами
      2.1.4. Таблица с читателями
3. **Логика (Presenter)**
   3.1. Класс реализации запросов
   3.2. Класс визуализации данных
   3.3. Класс взаимодействия с пользователем
      3.3.1. Логика окна «Главное меню»
      3.3.2. Логика окна «Каталог»
      3.3.3. Логика окна «Выдача книг»
      3.3.4. Логика окна «Работа с читателями»

После группировки получилась следующая структура директории приложения:

project-library

- database
  - library_db (2.1)

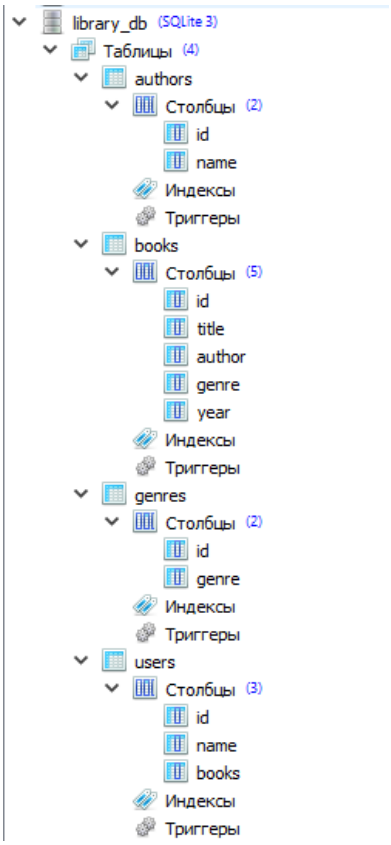
- scripts
  - book_widget_class (3.3.2)
  - give_widget_class (3.3.3)
  - users_widget_class (3.3.4)
  - db_class (3.1)
- UI
  - main_widget (1.1)
  - book_widget (1.1.1)
  - book_add_dialog (1.1.1.1.1)
  - book_change_dialog (1.1.1.1.2)
  - book_add_author_dialog (1.1.1.2.1)
  - book_change_author_dialog (1.1.1.2.2)
  - book_add_genre_dialog (1.1.1.3.1)
  - book_change_genre_dialog (1.1.1.3.2)
  - give_widget (1.1.2)
  - give_give_dialog (1.1.2.1)
  - give_back_dialog (1.1.2.2)
  - users_widget (1.1.3)
  - users_add_dialog (1.1.3.1)
  - users_change_dialog (1.1.3.2)
- main (3.3.1)



Визуализация зависимостей

3.4. База данных

В SQlite была создана база данных «library_db», в которую были заложены следующие таблицы:



1. **authors**

Таблица, содержащая ФИО авторов и их id для обращения по id в других таблицах

2. **genres**

Таблица, содержащая названия жанров и их id для обращения по id в других таблицах

3. **books**

Таблица, содержащая название книги, год, id жанра, id автора и id книги, для обращения по id в других таблицах

4. **users**

Таблица, содержащая имя пользователя, id книги, которая ему выдана (если книги на руках нет статус будет равен 0) и id пользователя, для обращения по id в других таблицах

Реализованные особенности базы данных:

- Для столбцов id использовалась функция автоинкрементного первичного ключа.
- Новым объектам таблиц уникальный id будет присваиваться автоматически.
- При удалении выданной книги статус читателя автоматически становиться 0.
- Предусмотрены ситуации для книг без авторов и жанров.
- При удалении автора или жанра книги, значения автора или жанра автоматически становится «не указан»

3.5.Графический интерфейс

Графический интерфейс программы состоит из 14 объектов, каждый из которых был создан в редакторе «QT designer», а именно:

1. **main_widget (главное меню)**

Окно с размерами по умолчанию 600x400 на котором расположена надпись «Приложение для библиотеки» и 3 кнопки сгруппированных вертикально (Каталог книг, выдача книг, работа с читателями)

2. **book_widget (Вкладка каталога)**

Окно с размерами по умолчанию 600x400 на котором расположена кнопка для возвращения в главное меню и 3 вкладки:

- Каталог книг

На данной вкладке расположены: поисковая строка с выбором режима поиска (по автору, году, названию, жанру); кнопка вызова диалога добавления книги; таблица, отображающая информацию о книгах; поясняющий текст

- Каталог авторов

На данной вкладке расположены: поисковая строка; кнопка вызова диалога добавления автора; таблица, отображающая информацию об авторах поясняющий текст

- Каталог жанров

На данной вкладке расположены: кнопка вызова диалога добавления жанра; таблица, отображающая информацию о жанрах; поясняющий текст

3. **book_add_dialog (Диалог добавления книги)**

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: строка для ввода названия книги; строка для ввода года; поле со списком авторов; поле со списком жанров; кнопки «Добавить» и «Отмена»; поясняющий текст

4. **book_change_dialog (Диалог редактирования книги)**

Диалоговое окно выполнено аналогично «book_add_dialog», в котором кнопки заменены на «Редактировать», «Удалить», «Отмена»

5. **book_add_author_dialog (Диалог добавления автора)**

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: строка для ввода ФИО автора; кнопки «Добавить» и «Отмена»; поясняющий текст

6. book_change_author_dialog (Диалог редактирования автора)

Диалоговое окно выполнено аналогично «book_add_author_dialog», в котором кнопки заменены на «Редактировать», «Удалить», «Отмена»

7. book_add_genre_dialog (Диалог добавления жанра)

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: строка для ввода названия жанра; кнопки «Добавить» и «Отмена»; поясняющий текст

8. book_change_genre_dialog (Диалог редактирования жанра)

Диалоговое окно выполнено аналогично «book_add_genre_dialog», в котором кнопки заменены на «Редактировать», «Удалить», «Отмена»

9. give_widget (Окно выдачи книг)

Окно с размерами по умолчанию 600x400 на котором расположены: кнопка для вызова диалога выдачи книги; кнопка для вызова диалога получения книги; кнопка для возвращения в главное меню; таблица со списком читателей с книгой на руках; поясняющий текст

10.give_give_dialog (Диалог регистрации выдачи книги)

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: строка для поиска читателей; таблица с читателями; поле со списком авторов; поле со списком названий книг; кнопки «Выдать» и «Отмена»; поясняющий текст

11.give_back_dialog (Диалог регистрации получения книги)

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: таблица с читателями; информационные поля; кнопки «Получить» и «Отмена»; поясняющий текст

12.users_widget (Окно работы с читателями)

Окно с размерами по умолчанию 600x400 на котором расположены: кнопка для вызова диалога добавления читателя; кнопка для возвращения в главное меню; таблица со списком читателей; поясняющий текст

13.users_add_dialog (Диалог добавления пользователя)

Диалоговое окно с фиксированными размерами – 400x257, на котором расположены: строка для ввода ФИО пользователя; кнопки «Добавить» и «Отмена»; поясняющий текст

14.users_change_dialog (диалог редактирования пользователя)

Диалоговое окно выполнено аналогично «users_add_dialog», в котором кнопки заменены на «Редактировать», «Удалить», «Отмена»

3.6.Программная часть

Код программы состоит из 5 частей, а именно:

1. **db_class**

Данный файл кода содержит класс «Data», использующий библиотеку sqlite3 и отвечающий за взаимодействие между базой данных и остальной программой. Объект класса при инициализации подключается к базе данных и представляет из себя набор из 25 функций-запросов в базу данных, которые можно классифицировать следующим образом:

- запросы для получения информации всей таблицы
- запросы для добавления строк в таблицах
- запросы для редактирования строк в таблицах
- запрос для удаления строк в таблицах
- запрос для получения информации о строке с конкретным ID

2. **book_widget_class**

Данный файл кода содержит класс «BookWidget», отвечающий за работу окна «каталог». Класс унаследован от “QWidget” библиотеки «PyQt». При инициализации объект класса принимает окно главного меню, для передачи флага об обновлении таблиц в других окнах, а также запускает функцию графического отображения окна. Класс состоит из функций, отвечающих за:

- Отображение таблиц
- Работу кнопок
- Вызов диалоговых окон

Код, отвечающий за логику каждого отдельного диалогового окна представляет из себя функцию, проверяющую корректность введенных данных, и при выполнении условий выполняет запрос на изменение/добавление/удаление ячейки таблицы через объект класса «Data», инициализированный в главном окне, делает вызов на обновление таблиц в других окнах

3. give_widget_class

Данный файл кода содержит класс «GiveWidget», отвечающий за работу окна «выдача книг». Унаследован от класса “QWidget” библиотеки «PyQt», построен по аналогии с классом «BookWidget»

4. users_widget_class

Данный файл кода содержит класс «UsersWidget», отвечающий за работу окна «Работа с читателями». Унаследован от класса “QWidget” библиотеки «PyQt», построен по аналогии с классом «BookWidget»

5. main

Данный файл кода содержит класс «Main», отвечающий за работу окна «главное меню», а также функцию для запуска приложения. Унаследован от класса “QWidget” библиотеки «PyQt». Основной особенностью класса является одновременная инициализация окон, расположенных каскадом, что позволяет не тратить время на отрисовку таблиц.

3.7.Тестирование программы

Оформив весь код и графический интерфейс, я приступил к устранению ошибок в работе программы. Во время тестирования были выявлены и устранены следующие ошибки:

- При изменении значений в одной вкладке приложения, соответствующие изменения не отображались в других вкладках. Это вызвано тем, что исполняемые файлы каждого окна никак друг с другом не взаимодействуют. Для устранения данной ошибки в связующий класс «Main» была добавлена функция, обновляющая все таблицы приложения
- При выполнении запроса на удаление в базе данных значения не устанавливались «по умолчанию», из-за этого пришлось вручную прописать команду на запуск опознания внешних ключей, хотя при инициализации базы данных данный параметр уже включен.

В итоге получилось приложение, способное выполнять следующие функции:

1. Хранить, добавлять, редактировать, удалять информацию о книгах
2. Хранить, добавлять, редактировать, удалять информацию об авторах книг
3. Хранить, добавлять, редактировать, удалять информацию о жанрах книг
4. Хранить, добавлять, редактировать, удалять информацию о пользователях
5. Изменять и хранить статус книг

Можно выделить следующие преимущества и особенности приложения:

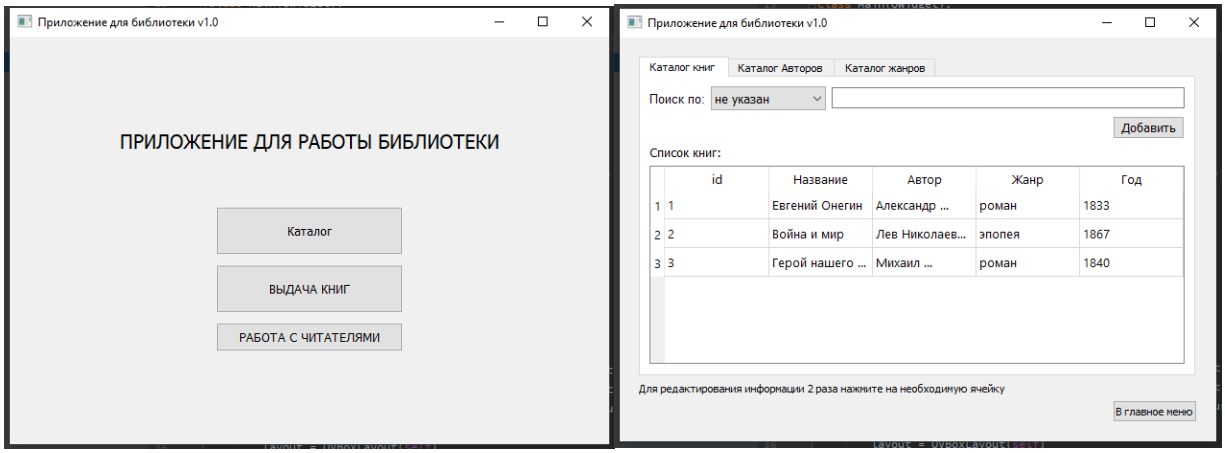
- Параллельная работа окон, расположенных каскадом
- База данных полностью обособлена от интерфейса
- Редактирование ячеек выполняется после двойного нажатия
- Изменения сохраняются в базе данных

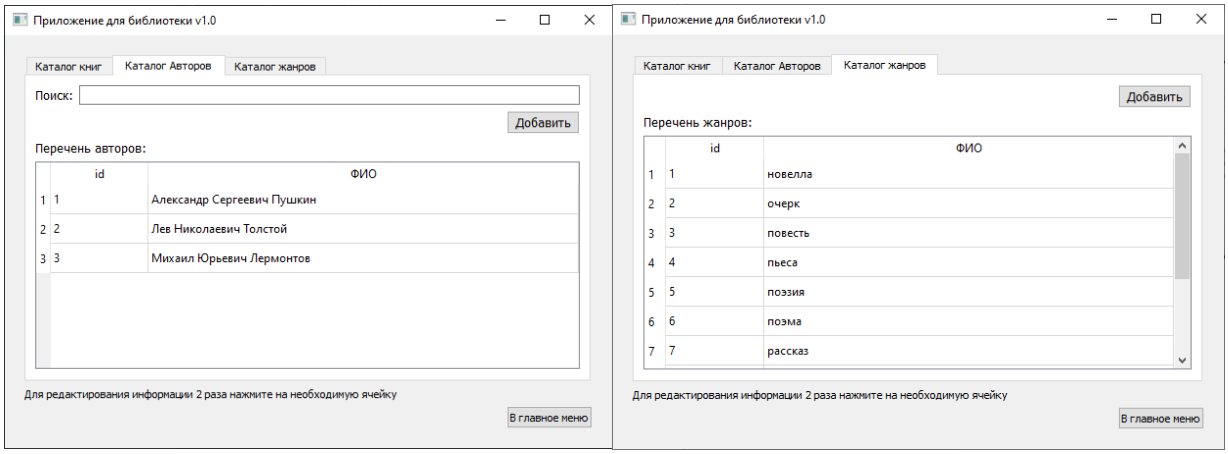
4.Заключение

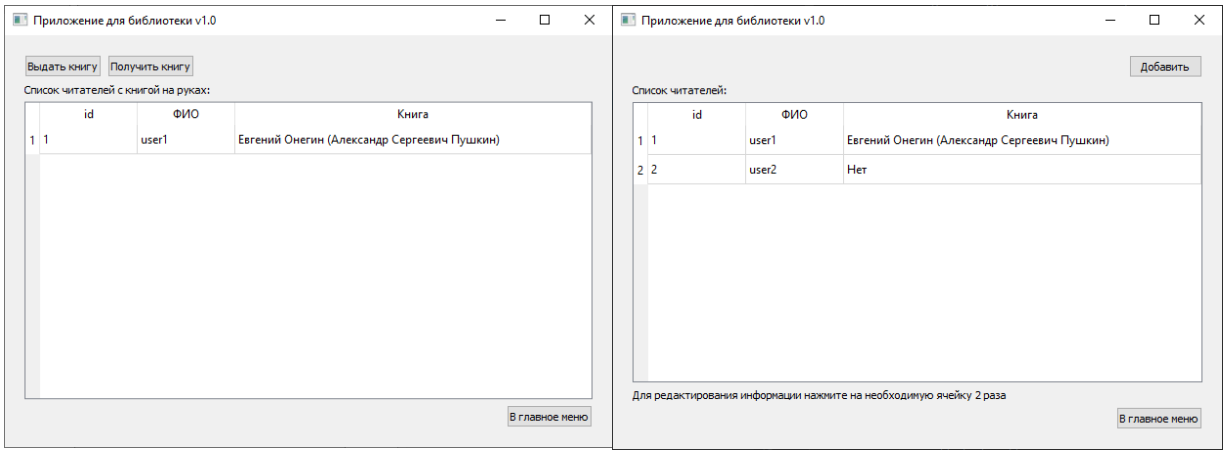
Современные технологии не стоят на месте, программное обеспечение и компьютеры заменяют уже устаревшие технологии, позволяя ускорить процесс их выполнения. Выполняя работы по созданию приложения для библиотеки и решения поставленной проблемы я узнал, как можно архитектурно строить программы, научился множеству новых приемов в программировании. В дальнейшем планирую интегрировать данное приложение в школьную библиотеку.

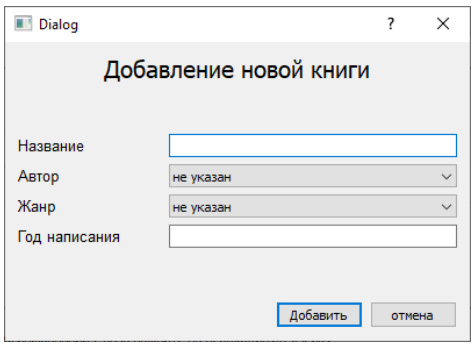
5.Приложения

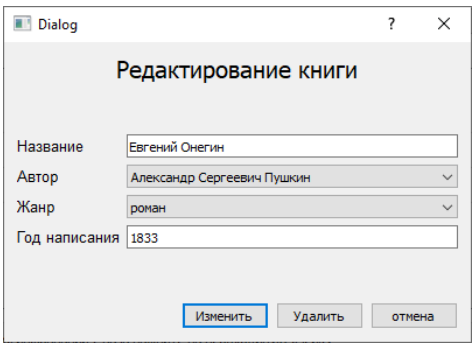
1) Графический интерфейс программы

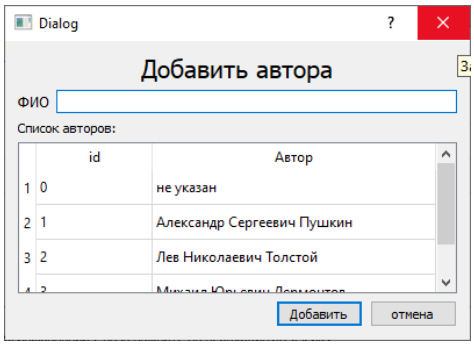


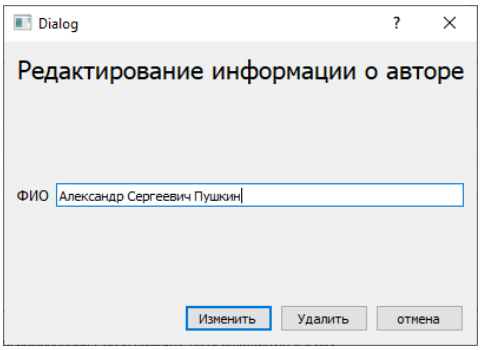


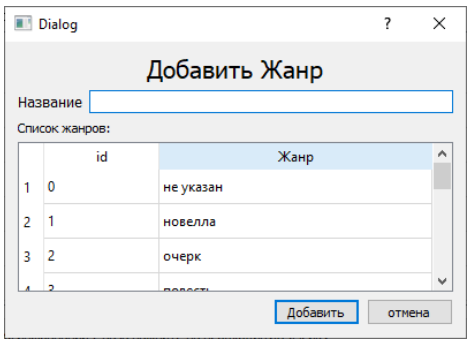


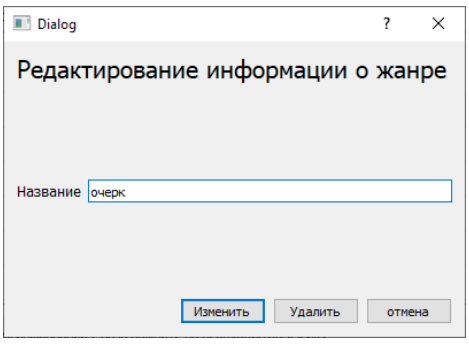

Dialog
Добавление новой книги
Название
Автор
не указан
Жанр
не указан
Год написания
Добавить
отмена

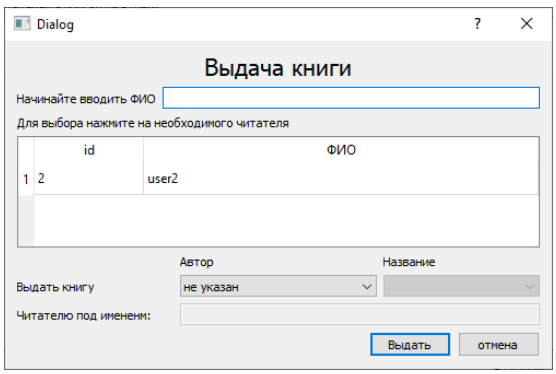

Dialog
Редактирование книги
Название
Евгений Онегин
Автор
Александр Сергеевич Пушкин
Жанр
роман
Год написания
1833
Изменить
Удалить
отмена

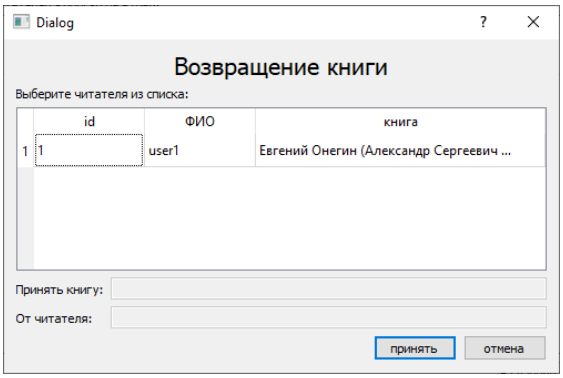

Dialog
Добавить автора
ФИО
Список авторов:
id
Автор
1 0 не указан
2 1 Александр Сергеевич Пушкин
3 2 Лев Николаевич Толстой
Добавить
отмена

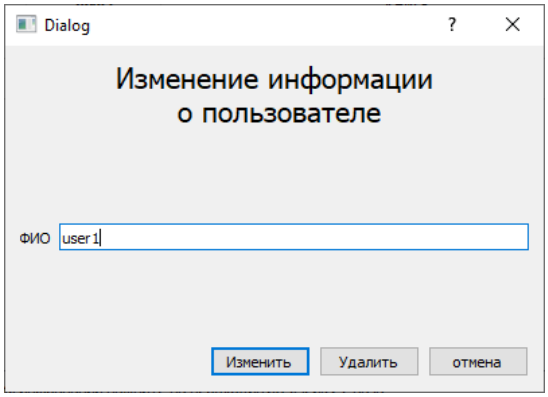

Dialog
Редактирование информации о авторе
ФИО
Александр Сергеевич Пушкин
Изменить
Удалить
отмена

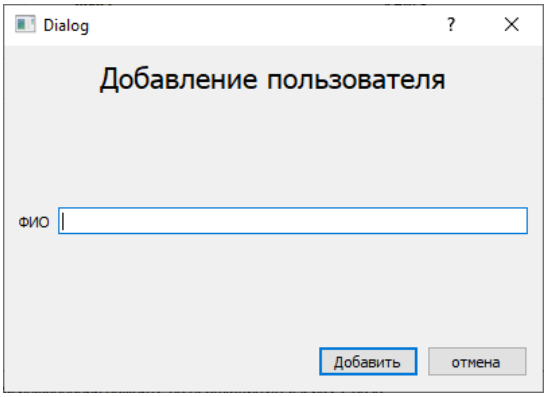

Dialog
Добавить Жанр
Название
Список жанров:
id
Жанр
1 0 не указан
2 1 новелла
3 2 очерк
Добавить
отмена


Dialog
Редактирование информации о жанре
Название
очерк
Изменить
Удалить
отмена


Dialog
Выдача книги
Начинайте вводить ФИО
Для выбора нажмите на необходимого читателя
id
ФИО
1 2 user2
Автор
Название
Выдать книгу
не указан
Читателю под именем:
Выдать
отмена


Dialog
Возвращение книги
Выберите читателя из списка:
id
ФИО
книга
1 1 user1 Евгений Онегин (Александр Сергеевич ...
Принять книгу:
От читателя:
принять
отмена


Dialog
Изменение информации
о пользователе
ФИО
user1
Изменить
Удалить
отмена


Dialog
Добавление пользователя
ФИО
Добавить
отмена

2) ссылка на репозиторий проекта: https://github.com/Charmpy/library-project