****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)»

филиал «РКТ» МАИ в г. Химки Московской области

**Специальность** 09.02.07 Информационные системы и программирование

**Группа** ИСП41-19

**Квалификация** Программист

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

По МДК 02.01 Технология разработки программного обеспечения

(указывается код и название дисциплины, МДК, ПМ)

На тему: Разработка информационной системы «Библиотека»

Автор курсового проекта (работы) Шепелёв Сергей Александрович /

(Фамилия Имя Отчество) (подпись)

Руководитель Жилина Татьяна Анатольевна /

(Фамилия Имя Отчество) (подпись)

****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)» (МАИ)

филиал «РКТ» МАИ в г. Химки Московской области

**Специальность** 09.02.07 Информационные системы и программирование

**Группа** ИСП41-19

**Квалификация** Программист

З А Д А Н И Е

**на курсовой проект (работу)**

По МДК 02.01 Технология разработки программного обеспечения

(указывается код и название дисциплины, МДК, ПМ)

**Студенту** Шепелёву Сергею Александровичу /

(Фамилия Имя Отчество) (подпись)

**Руководитель** Жилина Татьяна Анатольевна /

(Фамилия Имя Отчество) (подпись)

Преподаватель, филиал «РКТ» МАИ в г. Химки Московской области

(Ученая степень, ученое звание, должность и место работы)

**1. Наименование темы:** Разработка информационной системы «Библиотека» ㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤㅤ

**2. Срок сдачи студентом законченной работы**

**3. Техническое задание и исходные данные к работе**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Перечень подлежащих разработке разделов и этапы выполнения работы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | Наименование раздела или этапа | Трудоёмкость в % от  полной трудоёмкости  курсовой проекта(работы) | Срок выполнения | Примечание |
| 1 | 2 | 3 | 4 |  |
| 1 | Разработка технического задания | 10% | 17.11.2022 |  |
| 2 | Выбор структур данных, разработка иерархии объектов | 20% | 24.11.2022 |  |
| 3 | Разработка алгоритмов и реализация программы на выбранном языке программирования | 45% | 19.01.2023 |  |
| 4 | Подготовка расчетно-пояснительной записки | 10% | 26.01.2023 |  |
| 5 | Защита курсовой работы | 15% | 02.02.2023 |  |

**5. Перечень иллюстративно-графических материалов:**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Количество листов** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**6. Исходные материалы и пособия**

**7. Дата выдачи задания**  10.11.2022

Руководитель

(подпись)

Задание принял к исполнению

(подпись)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ………………………………………………………...…………………………….…....5](#_Toc119621095)

[1 Разработка системного проекта………………………………………………………………..….6](#_Toc98283917)

[1.1 Назначение разработки……………………………………………………………………….18](#_Toc98283919)

[1.2 Требования к функциональным характеристикам…………………………………………..19](#_Toc98283920)

[1.3 Требования к надёжности и безопасности…………………………………………………..24](#_Toc98283921)

[1.4 Требования к составу и параметрам технических средств…………………………………24](#_Toc98283921)

[1.5 Требования к информационной и программной совместимости…………………………..24](#_Toc98283921)

[2 Разработка технического проекта………………………………………………………………...18](#_Toc98283918)

[2.1 Анализ требований и определение спецификаций программного обеспечения…………..18](#_Toc98283919)

[2.2 Проектирование модели данных……………………………………………………………...19](#_Toc98283920)

[2.3 Детальное проектирование программного обеспечения…………………………………….24](#_Toc98283921)

[3 Реализация………………………………………………………………..........................................25](#_Toc98283922)

[3.1 Обоснование выбора средств разработки…………………………………….........................25](#_Toc98283923)

[3.2 Описание основных программных модулей………………………………………………….26](#_Toc98283924)

[4 Описание способа сборки программы…………………………………………………………….31](#_Toc98283925)

[5 Описание способа и методов взаимодействия с базой данных………………………………….36](#_Toc98283926)

[ЗАКЛЮЧЕНИЕ………………………………………………………………………………………..42](#_Toc98283928)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ…………………………………………………44](#_Toc98283929)

[Приложение А. Исходный код программы………………………………………………………….45](#_Toc98283930)

[Приложение Б. Исходный код функциональных тестов…………………………………………....67](#_Toc98283935)

# ВВЕДЕНИЕ

Библиотека – это культурное учреждение, в котором собираются, хранятся и предоставляются для общественного пользования бумажные носители информации – книги, журналы, газеты и так далее. Автоматизированная информационная система (ИС) «Библиотека» представляет собой автоматизированную систему учёта книг в библиотеке, поиска определённой книги, а также выдача её во временное пользование читателю.

Тема реализации ИС не нова в связи с тем, что в современном мире всё больше отдаётся предпочтение цифровым решениям, нежели традиционным, поэтому разработка такого рода программного продукта актуальна для всех учреждений, в том числе и для библиотеки. Обусловлено это тем, что большинство библиотек продолжают использовать бумажные системы учёта, и испытывают трудности в переходе на использование современных цифровых средств. В связи с этим необходимо предоставить библиотекам простое и удобное в использовании приложение, способствующие плавному переходу от традиционных способов учёта к цифровым. Исходя из этого, целью разработки является решение вопроса автоматизации формирования библиотечных списков выписанных для читателя книг, упрощения процесса упорядочивания состава книжного фонда библиотеки, а также облегчения учёта должников библиотеки.

Основными задачами курсовой работы является теоретическая подготовка к разработке программного продукта, проектирование базы данных по выбранной предметной области, разработка приложения с графическим интерфейсом, составление отчёта по итогам проделанной работы.

# Анализ инструментальных средств

Таблица 1 – Сравнительная характеристика языков программирования.

|  |  |  |
| --- | --- | --- |
| Язык программирования | Достоинства | Недостатки |
| C++ | 1. Поддержка объектно-ориентированного программирования (ООП); 2. Высокая производительность; 3. Возможности для работы с данными на низком уровне — то есть на уровне, близком к аппаратному; | 1. Небезопасность: С++ даёт большую свободу действий, но и не удержит вас от ошибок; 2. Зависимость от платформы: написать на С++ портативный код (такой, который бы работал на разных платформах) очень сложно; 3. Синтаксис строгий и «многословный»: код читается хуже, чем в некоторых других языках (например в Python); |
| Python | 1. Расширяемость и гибкость; 2. Интерпретируемость и кроссплатформенность; 3. Простой синтаксис. 4. Интерпретатор с открытым исходным кодом; | 1. Высокий уровень зависимости от системных библиотек; 2. Global Interpreter Lock не даёт возможности одновременно исполнять несколько потоков; |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Язык программирования | Достоинства | Недостатки |
| Java | 1. Использование в корпоративных приложениях, Java способен поддержать строительные блоки системы или различные библиотеки, с их помощью создают необходимые функции; 2. Запуск приложений в «песочнице» с устранением распространённых, уязвимых объектов в соответствии с политикой безопасности; 3. Независимое положение от платформ; 4. Повышенная производительность труда благодаря встроенному механизму, чтобы совместно использовать данные программы на нескольких компьютерах; | 1. Низкая производительность и безопасность; 2. Отсутствует нативный дизайн. Чтобы использовать интерфейс; ориентированный для языка нужно изучить каждый, выбрать шаблон иначе будет заметно несоответствие фрагментов; 3. Многословность и сложность кода; |

Таблица 2 – Сравнительная характеристика IDE

|  |  |  |
| --- | --- | --- |
| Среда разработки | Достоинства | Недостатки |
| PyCharm | 1. Простая организация проектов; 2. Автоматический рефакторинг при переименовании и удалении файлов; 3. Можно проводить интегрированное Unit тестирование; | 1. Некоторые функции доступны только в платной версии; 2. Может возникнуть проблема при попытке исправить такие инструменты, как venv; 3. IDE ресурсоёмкий; |
| MS Visual Studio | 1. Встроенный Web-сервер; 2. Поддержка множества языков при разработке; 3. Интуитивный стиль кодирования; 4. Высокая скорость разработки; | 1. Невозможность отладчика (Microsoft Visual Studio Debugger) отслеживать в коде режима ядра; 2. Тяжеловесная IDE, достаточно требовательная к аппаратному обеспечению; 3. Не работает на Linux; |
| Code::Blocks | 1. Кроссплатформенная среда разработки; 2. Распространяется бесплатно; 3. Легковесная среда разработки; | 1. Отсутствие возможности дебага при использовании MSVC компилятора; 2. Недостаточная функциональность для создания крупных проектов; 3. Нестабильность в работе; |

# Подготовка к разработке

## Диаграмма вариантов использования

Унифицированный язык моделирования (Unified Modeling Language, UML) – это графический язык для визуализации, специфицирования, конструирования и документирования систем, в которых глав-ная роль принадлежит программному обеспечению. С помощью UML можно разработать детальный план создаваемой системы, содержащий не только ее концептуальные элементы, такие как системные функции и бизнес процессы, но и конкретные особенности, например классы, написанные на каком-либо языке программирования, схемы баз данных и повторно используемые программные компоненты [2].

Диаграмма вариантов использования (Use Case Diagram) — диаграмма, на которой отражены отношения, существующие между акторами (действующими лицами) и вариантами использования (прецедентами). Основная задача — представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы. Основные элементы диаграммы – действующее лицо (actor) и прецедент (вариант).

* Участник – это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Участником может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности. Графически участник изображается “человечком”.
* Прецедент (use case) – описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому участником результату. Прецедент представляет поведение сущности, описывая взаимодействие между участниками и системой. Прецедент не показывает, “как” достигается некоторый результат, а только “что” именно выполняется. Прецеденты обозначаются очень простым образом - в виде эллипса, внутри которого указано его название.
* Добавить общ эллипс «авторизация»

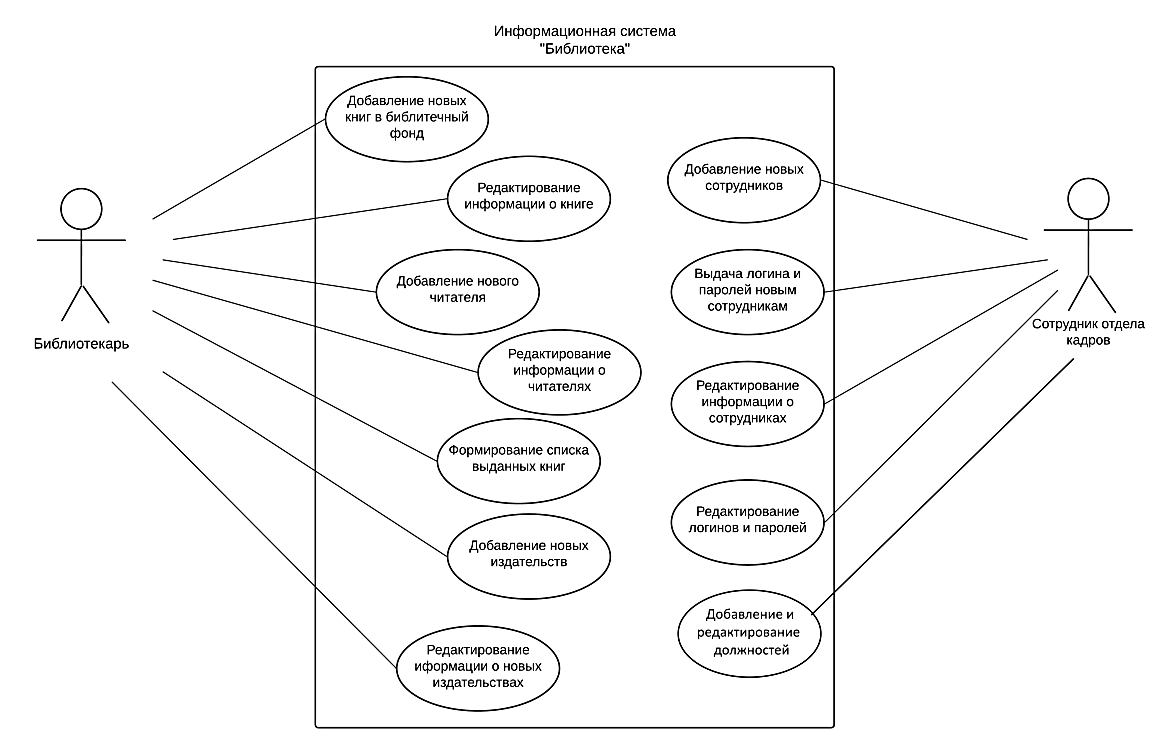


Рисунок 2 – UML-диаграмма

Структурная схема

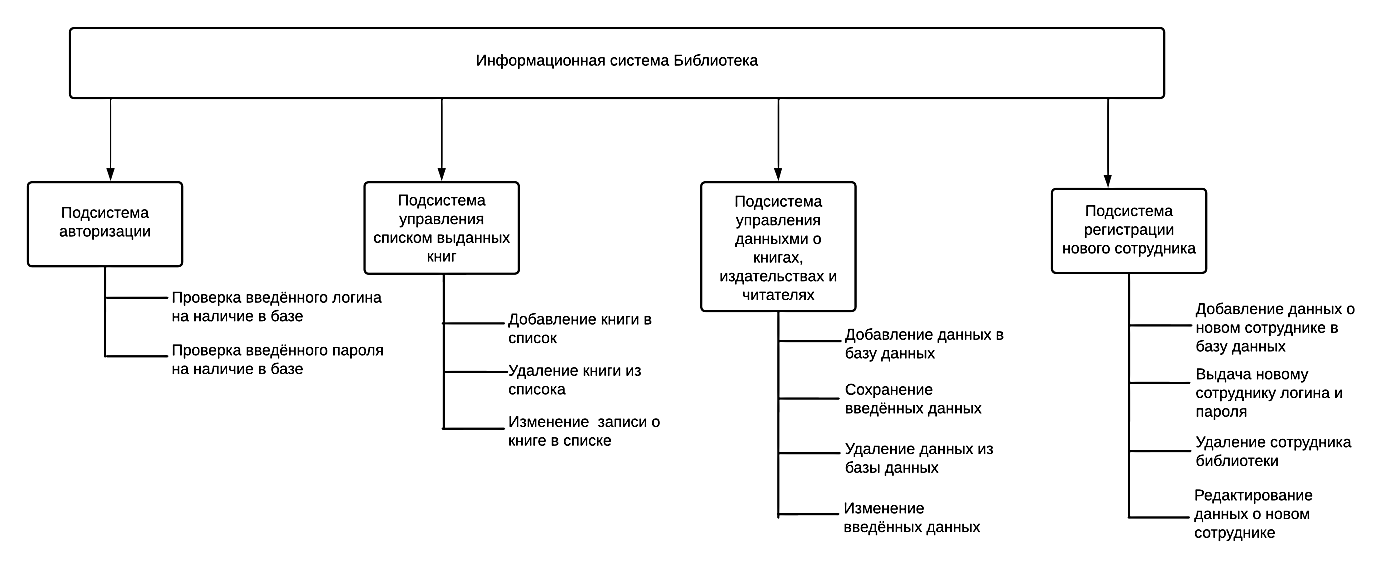


Рисунок 3 – Структурная схема

Функциональная схема(отредачить по образцу из листка)

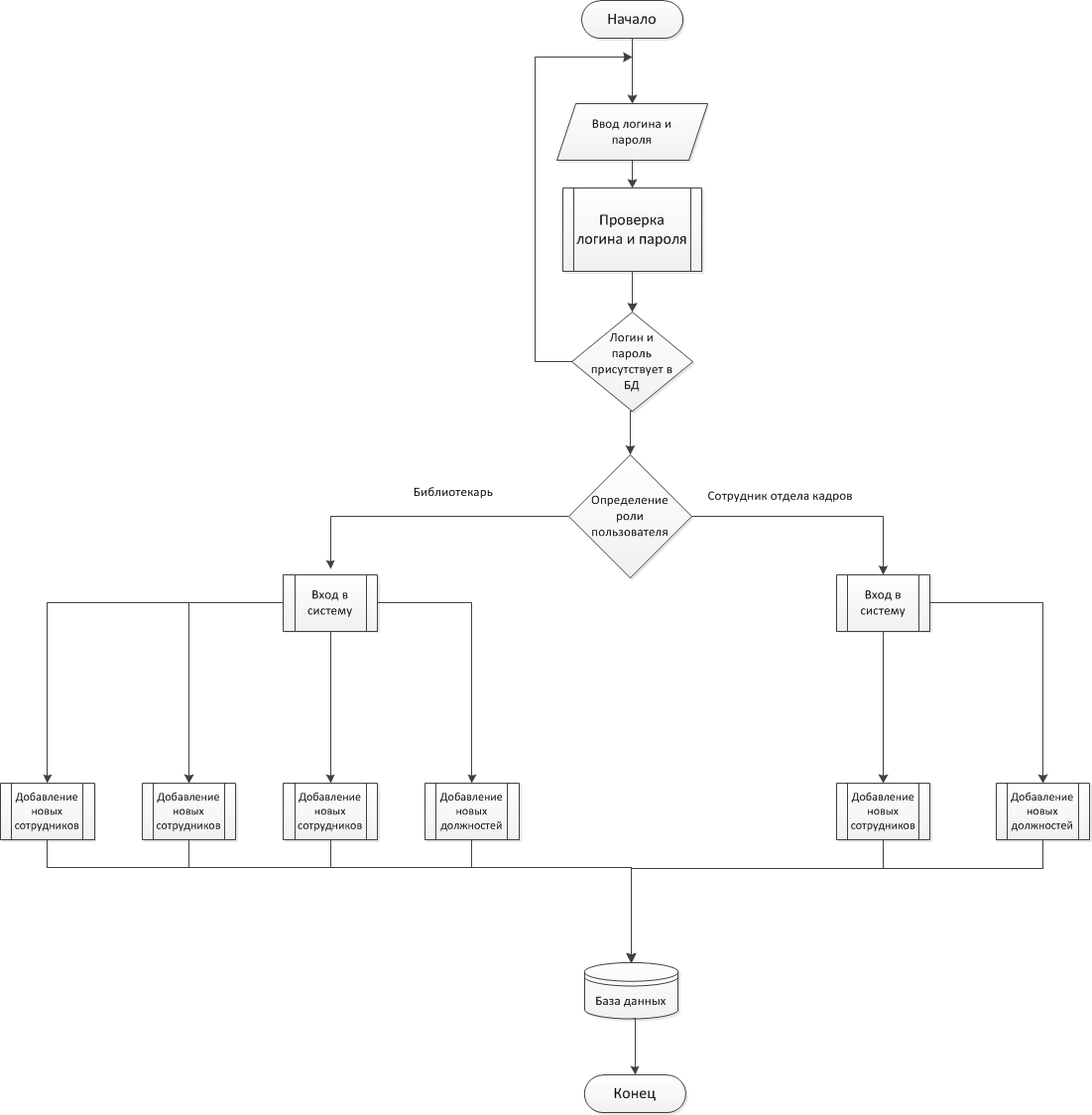


Рисунок 4 – Функциональная схема пользователя «Библиотекарь»

Функциональная диаграмма

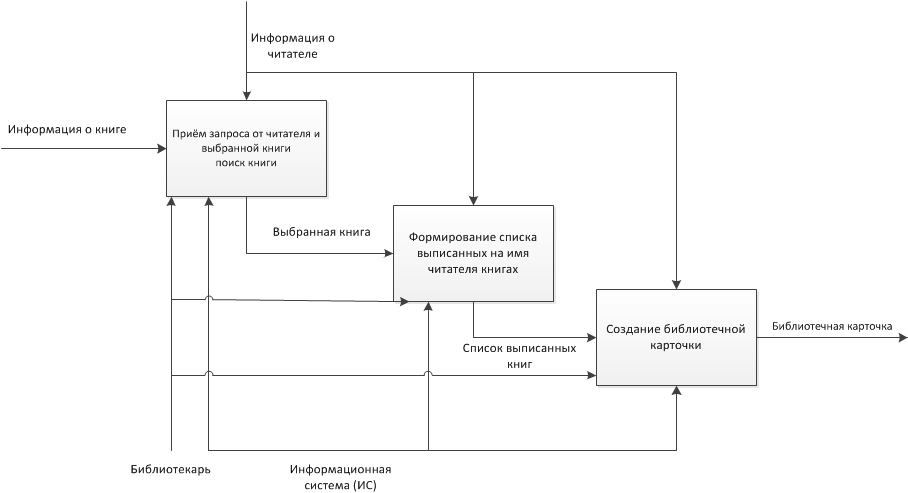


Рисунок 5 – Функциональная диаграмма ИС «Библиотека»

Диаграмма потоков данных

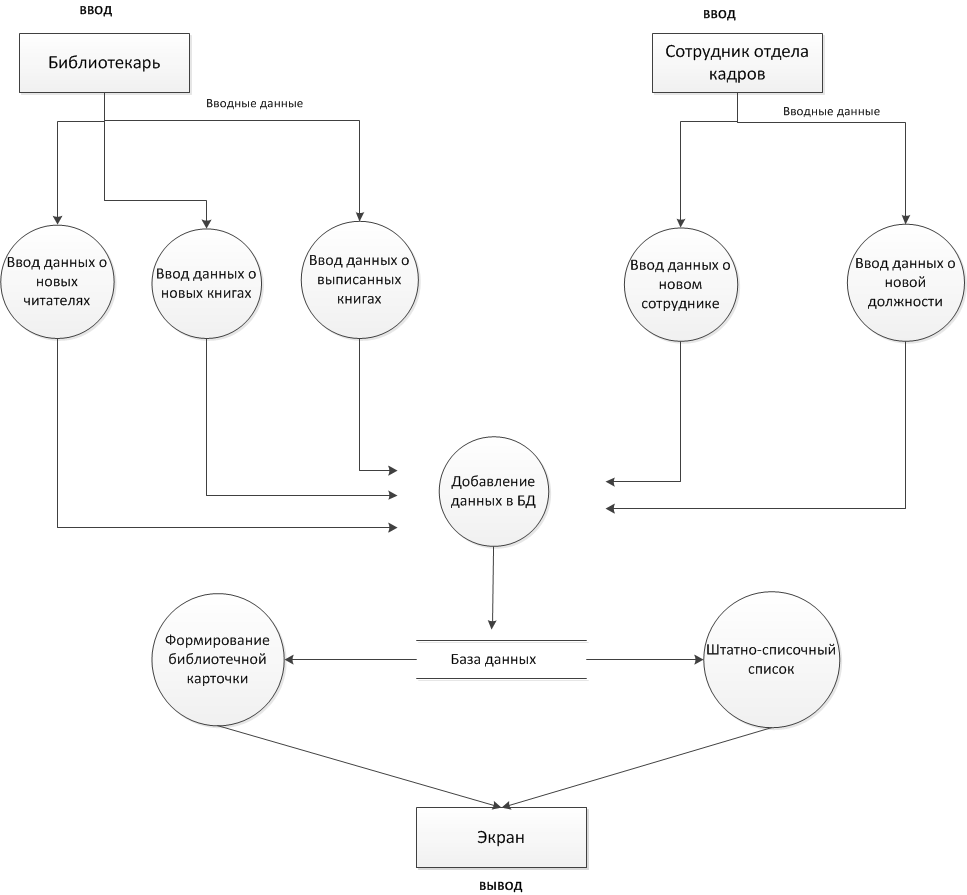


Рисунок 6 – Диаграмма потоков данных

Диаграмма деятельности

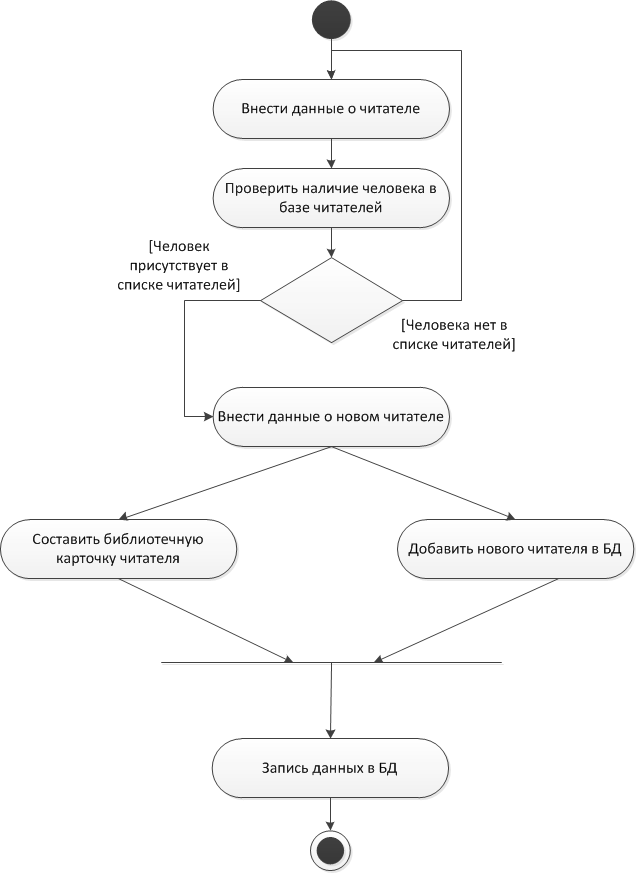


Рисунок 7 – Диаграмма деятельности

# 3 Разработка приложения

ER-модель данных обеспечивает стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Диаграммы «сущность— связь» в отличие от функциональных диаграмм определяют спецификации структур данных программного обеспечения.

Средствами Lucidchart была создана диаграмму «сущность-связь» для разрабатываемой программы, интерфейс ресурса представлен на рисунке 1. Ключевой особенность данного ресурса является возможность строить требуемые диаграммы и схемы без необходимости оформлять платную подписку. Интерфейс сайта прост и понятен. Слева находится панель с объектами, а справа поле для рисования.

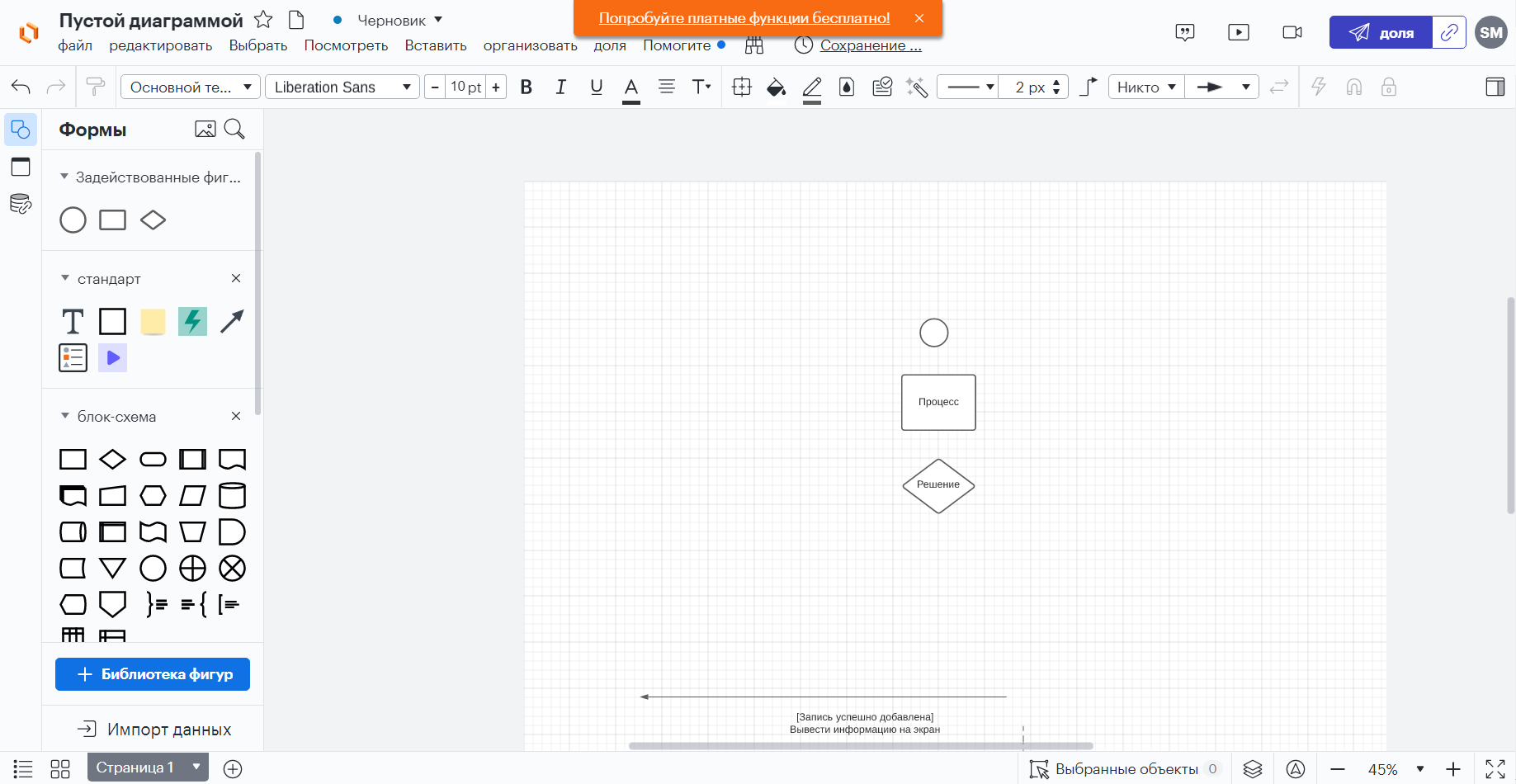


Рисунок 1 – Интерфейс Lucidchart.

Путём перетаскивания на поле можно отразить все ключевые схемы и диаграммы на итоговом рисунке. Помимо этого есть возможность соединить объекты между собой путём перетаскивания стрелок на промежуток между двумя объектами, либо посредством последовательного нажатия левого клика с одного объекта на другой.

В результате проделанных манипуляций была сделана диаграмма, представленная на рисунке 2.

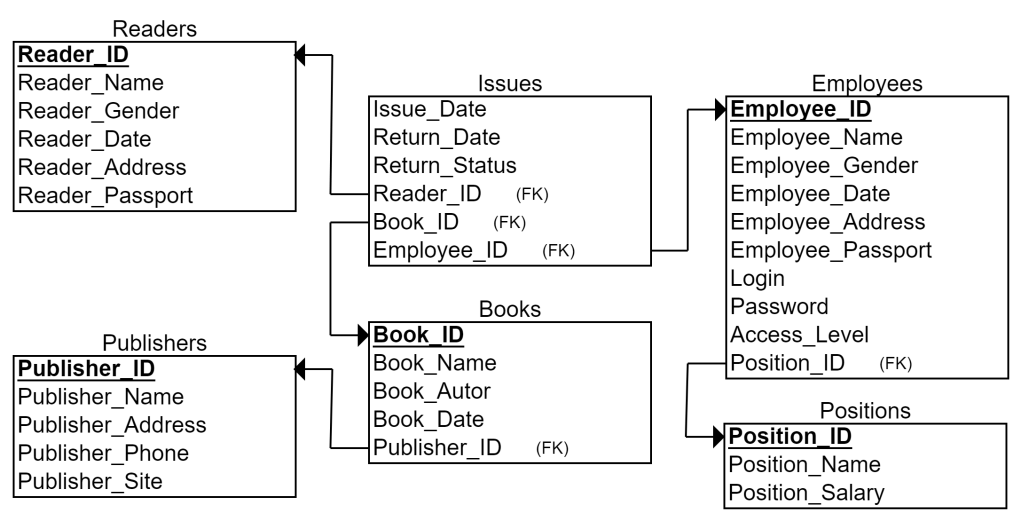


Рисунок 1 – Реляционная схема данных ИС «Библиотека»

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была достигнута её цель и решены следующие задачи:

1. Изучена литература, посвящённая разработке настольного приложения, проектированию, администрированию и защите базы данных;
2. Была спроектирована база данных в соответствии с рассматриваемой предметной областью;
3. Определены и разработаны программные модули необходимые для работы приложения. А именно:
   * Модуль, реализующий графический интерфейс программы, позволяющий удобно взаимодействовать с данными в базе данных, а также обладает возможностью отобразить изменения, вносимые пользователем в базу данных. Исходный код модуля представлен в приложении Б;
   * Модуль, реализующий запись, удаление и сохранение вводимых данных в базу данных. Исходный код модуля представлен в приложении Б.
4. Проведено тестирование программного модуля с целью проверки соответствия алгоритмам работы программы. Для этой цели был разработан модуль, реализующий функциональные тесты, необходимых для проверки программных модулей на работоспособность;
5. По результатам проделанной работы составлена документация к разработанному программному продукту.

В ходе написания курсовой роботы были закреплены основы в области разработки программных модулей. Результатом закрепленных знаний являются разработанные и корректно функционирующие программный продукт.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Изучаем программирование на Python / Пол Бэрри; [пер. с англ. М. А. Райтман]. – Москва: Издательство «Э», 2017. – 624 с.

2. Г. Буч, Д. Рамбо, И. Якобсон. Язык UML. Руководство пользователя

3. ГОСТ 7.32-2017 отчёт о научно-исследовательской работе. М., Стандартинформ, 2017.

# ПРИЛОЖЕННИЕ A. Иллюстрации окон приложения

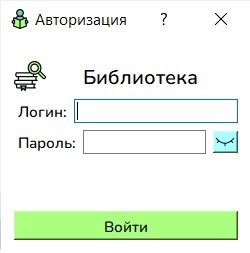


Рисунок А.1 – Окно авторизации

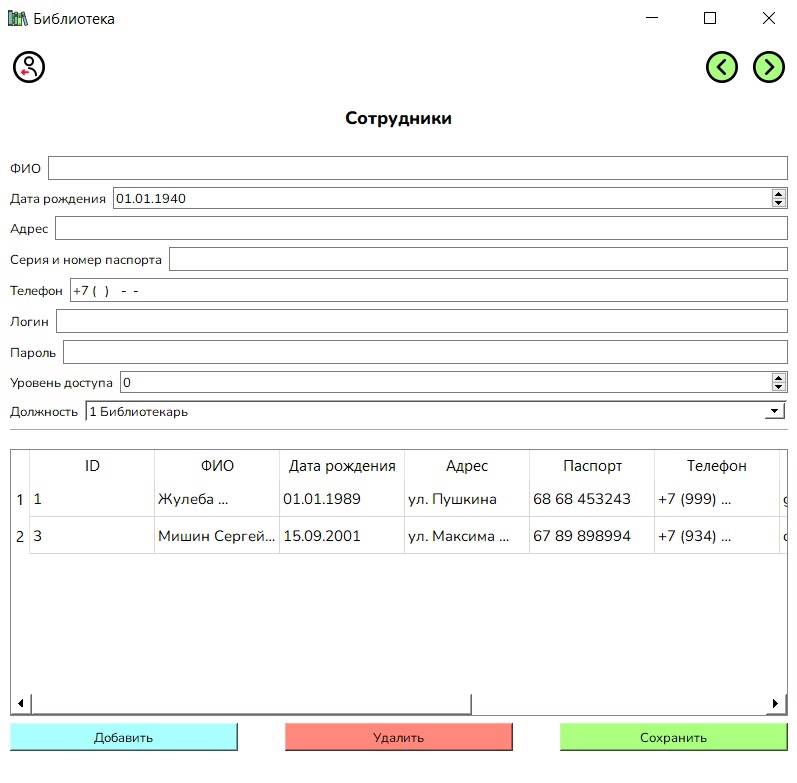


Рисунок А.2 – Страница сотрудников



Рисунок А.3 – Страница должностей



Рисунок А.4 – Страница издательств

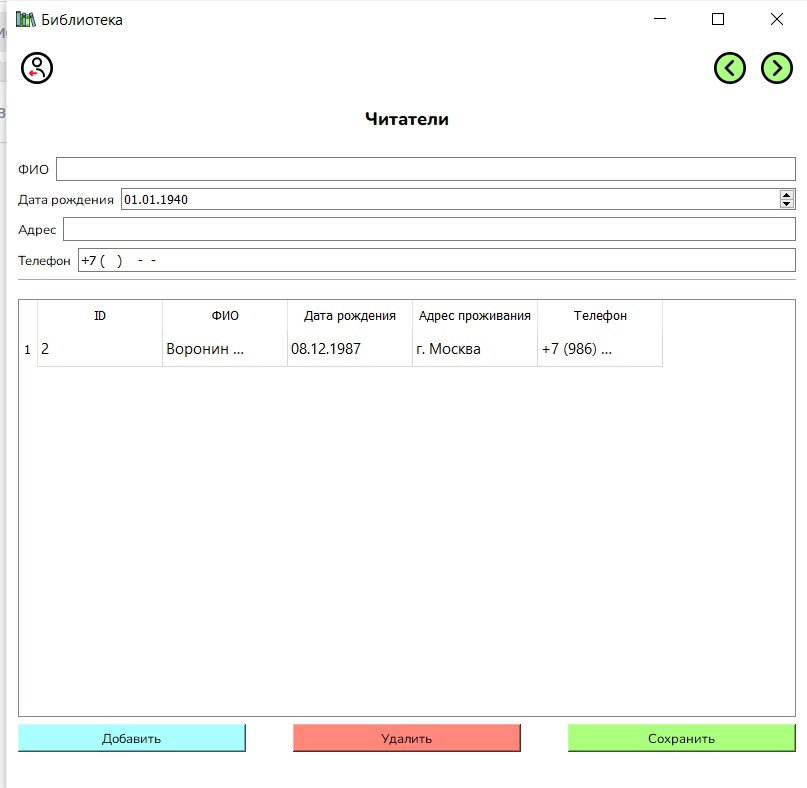


Рисунок А.5 – Страница читателей

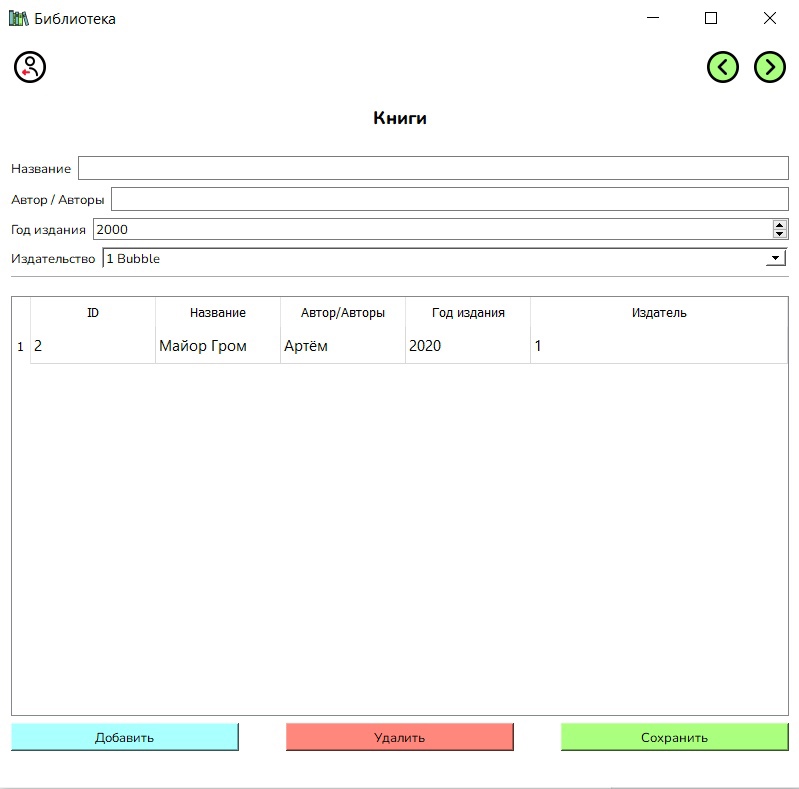


Рисунок А.6 – Страница книг

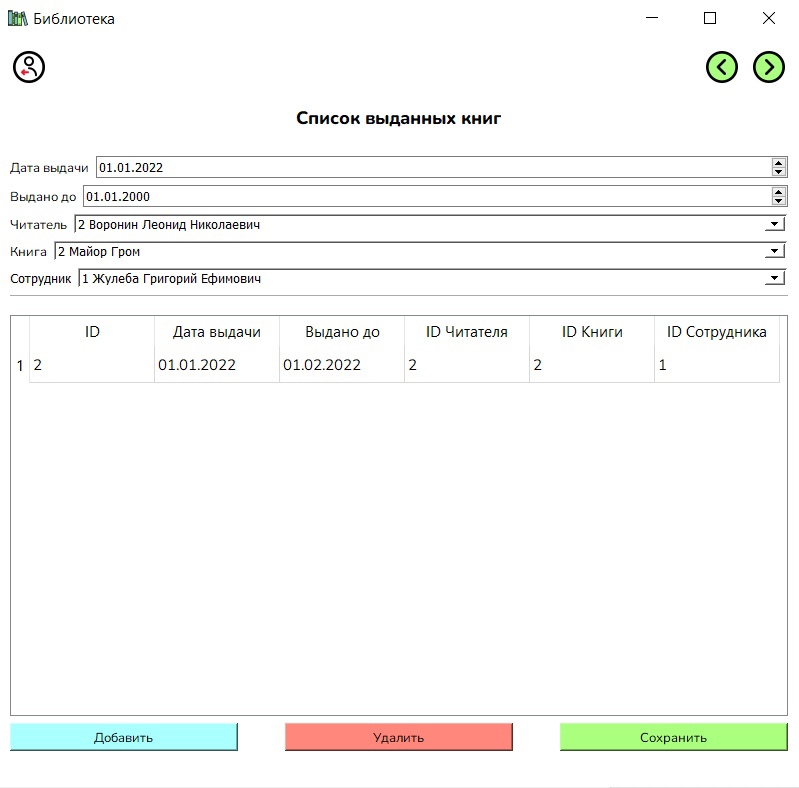


Рисунок А.7 – Список выданных книг

# ПРИЛОЖЕННИЕ Б. Исходный код приложения

**Б.1 Листинг кода “gui.py”**

from PyQt5.QtCore import Qt, QTime, QDateTime

from PyQt5.QtWidgets import QTableWidgetItem

from database import DataBase

import sys

from PyQt5 import QtWidgets

from PyQt5 import uic, QtCore, QtGui

from PyQt5.QtGui import QFont, QIcon

from PyQt5.QtWidgets import QApplication

from PyQt5.QtWidgets import QDialog

from PyQt5.QtWidgets import QGraphicsScene

from PyQt5.QtWidgets import QMainWindow

from PyQt5.QtWidgets import QMessageBox

import logging

logging.basicConfig(level=logging.INFO)

class MainWindow(QMainWindow):

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

super().\_\_init\_\_()

self.db = DataBase()

self.ui = uic.loadUi("forms/main.ui", self)

# self.exitButton.setIcon(QIcon('icons/logout.png'))

self.page = self.ui.stackedWidget\_main

self.page\_id = [0] # индексы доступных страничек после авторизации для сотрудника

self.now\_page = 0

self.ui.nextButton.clicked.connect(self.next\_page)

self.ui.nextButton.setToolTip("Следующая страница")

self.ui.backButton.clicked.connect(self.back\_page)

self.ui.backButton.setToolTip("Предыдущая страница")

self.ui.exitButton.clicked.connect(self.exit)

self.ui.exitButton.setToolTip("Завершить текущую сессию")

# self.ui.infoButton.setToolTip("Автор: Шепелёв Сергей, ИСП-41-19")

# Кнопки монипуляции данных таблицы Employees

self.ui.add\_emloyee.clicked.connect(self.new\_employee)

self.ui.delete\_emloyee.clicked.connect(self.delete\_employee)

self.ui.save\_emloyee.clicked.connect(self.save\_employee)

# Кнопки монипуляции данных таблицы Positions

self.ui.add\_pos.clicked.connect(self.new\_positions)

self.ui.delete\_pos.clicked.connect(self.delete\_position)

self.ui.save\_pos.clicked.connect(self.save\_position)

# Кнопки монипуляции данных таблицы Readers

self.ui.emp\_add.clicked.connect(self.new\_readers)

self.ui.emp\_delete.clicked.connect(self.delete\_reader)

self.ui.emp\_save.clicked.connect(self.save\_reader)

# Кнопки монипуляции данных таблицы Books

self.ui.add\_dep.clicked.connect(self.new\_books)

self.ui.delete\_dep.clicked.connect(self.delete\_book)

self.ui.save\_dep.clicked.connect(self.save\_book)

# Кнопки монипуляции данных таблицы Publishers

self.ui.pushButton\_2.clicked.connect(self.new\_publishers)

self.ui.pushButton\_3.clicked.connect(self.delete\_publisher)

self.ui.pushButton.clicked.connect(self.save\_publisher)

# Кнопки монипуляции данных таблицы Issues

self.ui.issues\_add.clicked.connect(self.new\_issues)

self.ui.issues\_delete.clicked.connect(self.delete\_issue)

self.ui.issues\_save.clicked.connect(self.save\_issue)

self.positions\_combobox.addItems(self.db.create\_combobox\_positions())

self.readers\_combobox.addItems(self.db.create\_combobox\_readers())

self.books\_combobox.addItems(self.db.create\_combobox\_books())

self.publishers\_combobox.addItems(self.db.create\_combobox\_publishers())

self.employee\_combobox.addItems(self.db.create\_combobox\_employees())

self.updateTableEmployees()

self.updateTablePositions()

self.updateTableBooks()

self.updateTableReaders()

self.updateTablePublishers()

self.updateTableIssues()

logging.log(logging.INFO, 'Приложение запущено.')

def exit(self):

self.now\_page = 0

self.page.setCurrentIndex(self.page\_id[self.now\_page])

self.hide()

logging.log(logging.INFO, 'Завершение сессии.')

self.open\_auth()

def next\_page(self):

if self.now\_page != len(self.page\_id) - 1:

self.now\_page += 1

self.page.setCurrentIndex(self.page\_id[self.now\_page])

logging.log(logging.INFO, 'Следующая страница.')

def back\_page(self):

if self.now\_page != 0:

self.now\_page -= 1

self.page.setCurrentIndex(self.page\_id[self.now\_page])

logging.log(logging.INFO, 'Предыдущая страница.')

def open\_auth(self):

dialog = DialogAuth(self)

dialog.setWindowTitle('Авторизация')

self.setWindowIcon(QIcon('icons/books.png'))

dialog.show()

dialog.exec\_()

logging.log(logging.INFO, 'Начало сессии.')

#########################

def updateTableEmployees(self):

self.table\_employees.clear()

rec = self.db.get\_from\_employees()

self.ui.table\_employees.setColumnCount(10)

self.ui.table\_employees.setRowCount(len(rec))

self.ui.table\_employees.setHorizontalHeaderLabels(

['ID', 'ФИО', 'Дата рождения', 'Адрес', 'Паспорт', 'Телефон', 'Логин', 'Пароль', 'Уровень доступа', 'ID должности'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_employees.setItem(i, x, item)

def updateTablePositions(self):

self.table\_positions.clear()

rec = self.db.get\_from\_positions()

self.ui.table\_positions.setColumnCount(3)

self.ui.table\_positions.setRowCount(len(rec))

self.ui.table\_positions.setHorizontalHeaderLabels(

['ID', 'Наименование', 'Оклад(₽)'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_positions.setItem(i, x, item)

def updateTableReaders(self):

self.table\_readers.clear()

rec = self.db.get\_from\_readers()

self.ui.table\_readers.setColumnCount(5)

self.ui.table\_readers.setRowCount(len(rec))

self.ui.table\_readers.setHorizontalHeaderLabels(

['ID', 'ФИО', 'Дата рождения', 'Адрес проживания', 'Телефон'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_readers.setItem(i, x, item)

def updateTableBooks(self):

self.table\_books.clear()

rec = self.db.get\_from\_books()

self.ui.table\_books.setColumnCount(5)

self.ui.table\_books.setRowCount(len(rec))

self.ui.table\_books.setHorizontalHeaderLabels(

['ID', 'Название', 'Автор/Авторы', 'Год издания', 'Издатель'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_books.setItem(i, x, item)

def updateTablePublishers(self):

self.table\_publishers.clear()

rec = self.db.get\_from\_publishers()

self.ui.table\_publishers.setColumnCount(5)

self.ui.table\_publishers.setRowCount(len(rec))

self.ui.table\_publishers.setHorizontalHeaderLabels(

['ID', 'Наименование', 'Адрес', 'Телефон', 'Сайт'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_publishers.setItem(i, x, item)

def updateTableIssues(self):

self.table\_issues.clear()

rec = self.db.get\_from\_issues()

self.ui.table\_issues.setColumnCount(6)

self.ui.table\_issues.setRowCount(len(rec))

self.ui.table\_issues.setHorizontalHeaderLabels(

['ID', 'Дата выдачи', 'Выдано до', 'ID Читателя', 'ID Книги', 'ID Сотрудника'])

for i, exposition in enumerate(rec):

for x, field in enumerate(exposition):

item = QTableWidgetItem()

item.setText(str(field))

if x == 0:

item.setFlags(Qt.ItemIsEnabled)

self.ui.table\_issues.setItem(i, x, item)

def getFromTableEmployees(self):

rows = self.table\_employees.rowCount()

cols = self.table\_employees.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_employees.item(row, col).text())

data.append(tmp)

return data

def getFromTablePositions(self):

rows = self.table\_positions.rowCount()

cols = self.table\_positions.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_positions.item(row, col).text())

data.append(tmp)

return data

def getFromTableBooks(self):

rows = self.table\_books.rowCount()

cols = self.table\_books.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_books.item(row, col).text())

data.append(tmp)

return data

def getFromTableReaders(self):

rows = self.table\_readers.rowCount()

cols = self.table\_readers.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_readers.item(row, col).text())

data.append(tmp)

return data

def getFromTablePublishers(self):

rows = self.table\_publishers.rowCount()

cols = self.table\_publishers.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_publishers.item(row, col).text())

data.append(tmp)

return data

def getFromTableIssues(self):

rows = self.table\_issues.rowCount()

cols = self.table\_issues.columnCount()

data = []

for row in range(rows):

tmp = []

for col in range(cols):

tmp.append(self.table\_issues.item(row, col).text())

data.append(tmp)

return data

def new\_employee(self):

employees\_fio = self.ui.employees\_fio.text()

employees\_date = self.ui.employees\_date.text()

employees\_address = self.ui.employees\_address.text()

employees\_passport = self.ui.employees\_passport.text()

employees\_phone = self.ui.employees\_phone.text()

login = self.ui.login.text()

password = self.ui.password.text()

access = self.ui.access.text()

self.ui.label\_9.setToolTip("0 - для библиотекаря, 1 - для сотрудника отдела кадров, 2 - для остальных должностей")

positions\_combobox = self.ui.positions\_combobox.currentText()

positions\_combobox = positions\_combobox.split(' ')[0]

if employees\_fio == '' or positions\_combobox == '' or password == '' or login == '' or access == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_employees(employees\_fio, employees\_date, employees\_address, employees\_passport, employees\_phone, login, password, access, positions\_combobox)

self.update\_combobox\_employees()

self.updateTableEmployees()

logging.log(logging.INFO, 'Запись добавлена.')

def new\_positions(self):

add\_pos\_name = self.ui.add\_pos\_name.text()

add\_pos\_salary = self.ui.add\_pos\_salary.text()

if add\_pos\_name == '' or add\_pos\_salary == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_positions(add\_pos\_name, add\_pos\_salary)

self.update\_combobox\_positions()

self.updateTablePositions()

logging.log(logging.INFO, 'Запись добавлена.')

def new\_readers(self):

add\_fio\_reader = self.ui.add\_fio\_reader.text()

add\_date\_birth\_reader = self.ui.add\_date\_birth\_reader.text()

add\_address\_reader = self.ui.add\_address\_reader.text()

add\_phone\_reader = self.ui.add\_phone\_reader.text()

if add\_fio\_reader == '' or add\_date\_birth\_reader == '' or add\_address\_reader == '' or add\_phone\_reader == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_readers(add\_fio\_reader, add\_date\_birth\_reader, add\_address\_reader, add\_phone\_reader)

self.update\_combobox\_readers()

self.updateTableReaders()

logging.log(logging.INFO, 'Запись добавлена.')

def new\_publishers(self):

lineEdit\_2 = self.ui.lineEdit\_2.text()

lineEdit\_3 = self.ui.lineEdit\_3.text()

lineEdit\_4 = self.ui.lineEdit\_4.text()

lineEdit\_5 = self.ui.lineEdit\_5.text()

if lineEdit\_2 == '' or lineEdit\_3 == '' or lineEdit\_4 == '' or lineEdit\_5 == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_publishers(lineEdit\_2, lineEdit\_3, lineEdit\_4, lineEdit\_5)

self.update\_combobox\_publishers()

self.updateTablePublishers()

logging.log(logging.INFO, 'Запись добавлена.')

def new\_books(self):

dep\_name\_line = self.ui.dep\_name\_line.text()

percent\_line = self.ui.percent\_line.text()

dateEdit = self.ui.dateEdit.text()

publishers\_combobox = self.ui.publishers\_combobox.currentText()

publishers\_combobox = publishers\_combobox.split(' ')[0]

if dep\_name\_line == '' or publishers\_combobox == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_books(dep\_name\_line, percent\_line, dateEdit, publishers\_combobox)

self.update\_combobox\_books()

self.updateTableBooks()

logging.log(logging.INFO, 'Запись добавлена.')

def new\_issues(self):

issues\_date = self.ui.issues\_date.text()

issues\_status = self.ui.issues\_status.text()

readers\_combobox = self.ui.readers\_combobox.currentText()

readers\_combobox = readers\_combobox.split(' ')[0]

books\_combobox = self.ui.books\_combobox.currentText()

books\_combobox = books\_combobox.split(' ')[0]

employee\_combobox = self.ui.employee\_combobox.currentText()

employee\_combobox = employee\_combobox.split(' ')[0]

if issues\_date == '' or readers\_combobox == '' or books\_combobox == '' or employee\_combobox == '':

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Заполните все поля")

msg.setWindowTitle("Ошибка")

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

logging.log(logging.INFO, 'Ошибка.')

else:

self.db.add\_in\_issues(issues\_date, issues\_status, readers\_combobox, books\_combobox, employee\_combobox)

self.updateTableIssues()

logging.log(logging.INFO, 'Запись добавлена.')

def update\_combobox\_positions(self):

self.positions\_combobox.clear()

self.positions\_combobox.addItems(self.db.create\_combobox\_positions())

logging.log(logging.INFO, 'Виджет ComboBox обновлён.')

def update\_combobox\_books(self):

self.books\_combobox.clear()

self.books\_combobox.addItems(self.db.create\_combobox\_books())

logging.log(logging.INFO, 'Виджет ComboBox обновлён.')

def update\_combobox\_readers(self):

self.readers\_combobox.clear()

self.readers\_combobox.addItems(self.db.create\_combobox\_readers())

logging.log(logging.INFO, 'Виджет ComboBox обновлён.')

def update\_combobox\_employees(self):

self.employee\_combobox.clear()

self.employee\_combobox.addItems(self.db.create\_combobox\_employees())

logging.log(logging.INFO, 'Виджет ComboBox обновлён.')

def update\_combobox\_publishers(self):

self.publishers\_combobox.clear()

self.publishers\_combobox.addItems(self.db.create\_combobox\_publishers())

logging.log(logging.INFO, 'Виджет ComboBox обновлён.')

def delete\_employee(self):

SelectedRow = self.table\_employees.currentRow()

rowcount = self.table\_employees.rowCount()

colcount = self.table\_employees.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_employees.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_employees.model().index(-1, -1)

self.table\_employees.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def delete\_position(self):

SelectedRow = self.table\_positions.currentRow()

rowcount = self.table\_positions.rowCount()

colcount = self.table\_positions.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_positions.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_positions.model().index(-1, -1)

self.table\_positions.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def delete\_book(self):

SelectedRow = self.table\_books.currentRow()

rowcount = self.table\_books.rowCount()

colcount = self.table\_books.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_books.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_books.model().index(-1, -1)

self.table\_books.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def delete\_reader(self):

SelectedRow = self.table\_readers.currentRow()

rowcount = self.table\_readers.rowCount()

colcount = self.table\_readers.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_readers.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_readers.model().index(-1, -1)

self.table\_readers.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def delete\_publisher(self):

SelectedRow = self.table\_publishers.currentRow()

rowcount = self.table\_publishers.rowCount()

colcount = self.table\_publishers.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_publishers.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_publishers.model().index(-1, -1)

self.table\_publishers.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def delete\_issue(self):

SelectedRow = self.table\_issues.currentRow()

rowcount = self.table\_issues.rowCount()

colcount = self.table\_issues.columnCount()

if rowcount == 0:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("В таблице нет данных!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

elif SelectedRow == -1:

msg = QMessageBox(self)

msg.setIcon(QMessageBox.Warning)

msg.setText("Выберите поле для удаления!")

msg.setWindowTitle("Ошибка")

logging.log(logging.INFO, 'Ошибка!')

msg.setStandardButtons(QMessageBox.Ok)

msg.exec\_()

else:

for col in range(1, colcount):

self.table\_issues.setItem(SelectedRow, col, QTableWidgetItem(''))

ix = self.table\_issues.model().index(-1, -1)

self.table\_issues.setCurrentIndex(ix)

logging.log(logging.INFO, 'Запись удалена.')

def save\_employee(self):

data = self.getFromTableEmployees()

for string in data:

if string[1] != '':

self.db.update\_employees(int(string[0]), string[1], string[2], string[3], string[4], string[5], string[6], string[7], int(string[8]), int(string[9]))

else:

self.db.delete\_from\_employees(int(string[0]))

self.update\_combobox\_employees()

self.updateTableEmployees()

logging.log(logging.INFO, 'Данные успешно записаны.')

def save\_position(self):

data = self.getFromTablePositions()

for string in data:

if string[1] != '':

self.db.update\_positions(int(string[0]), string[1], int(string[2]))

else:

self.db.delete\_from\_positions(int(string[0]))

self.update\_combobox\_positions()

self.updateTablePositions()

logging.log(logging.INFO, 'Данные успешно записаны.')

def save\_book(self):

data = self.getFromTableBooks()

for string in data:

if string[1] != '':

self.db.update\_books(int(string[0]), string[1], string[2], string[3], string[4])

else:

self.db.delete\_from\_books(int(string[0]))

self.update\_combobox\_books()

self.updateTableBooks()

logging.log(logging.INFO, 'Данные успешно записаны.')

def save\_reader(self):

data = self.getFromTableReaders()

for string in data:

if string[1] != '':

self.db.update\_readers(int(string[0]), string[1], string[2], string[3], string[4])

else:

self.db.delete\_from\_readers(int(string[0]))

self.update\_combobox\_readers()

self.updateTableReaders()

logging.log(logging.INFO, 'Данные успешно записаны.')

def save\_publisher(self):

data = self.getFromTablePublishers()

for string in data:

if string[1] != '':

self.db.update\_publishers(int(string[0]), string[1], string[2], string[3], string[4])

else:

self.db.delete\_from\_publishers(int(string[0]))

self.update\_combobox\_publishers()

self.updateTablePublishers()

logging.log(logging.INFO, 'Данные успешно записаны.')

def save\_issue(self):

data = self.getFromTableIssues()

for string in data:

if string[1] != '':

self.db.update\_issues(int(string[0]), string[1], string[2], int(string[3]), int(string[4]), int(string[5]))

else:

self.db.delete\_from\_issues(int(string[0]))

self.updateTableIssues()

logging.log(logging.INFO, 'Данные успешно записаны.')

class DialogAuth(QDialog):

def \_\_init\_\_(self, parent=None):

super(DialogAuth, self).\_\_init\_\_(parent)

self.ui = uic.loadUi("forms/auth.ui", self)

self.setWindowIcon(QIcon('icons/reading-book.png'))

self.scene = QGraphicsScene(0, 0, 300, 80)

self.ui.btn\_enter.clicked.connect(self.enter)

self.ui.btn\_enter.setToolTip("Войти в учётную запись")

self.btn\_hide\_password.setIcon(QIcon('icons/eye\_close.png'))

self.ui.btn\_hide\_password.clicked.connect(self.vis\_pas)

self.ui.btn\_hide\_password.setToolTip("Показать пароль")

self.db = DataBase()

self.next\_try = 0

self.vis\_p = False

def vis\_pas(self):

ed = self.ui.edit\_password

if self.vis\_p:

self.vis\_p = False

self.btn\_hide\_password.setIcon(QIcon('icons/eye\_close.png'))

self.ui.btn\_hide\_password.setToolTip("Показать пароль")

ed.setEchoMode(QtWidgets.QLineEdit.Password)

else:

self.vis\_p = True

self.btn\_hide\_password.setIcon(QIcon('icons/eye.png'))

self.ui.btn\_hide\_password.setToolTip("Скрыть пароль")

ed.setEchoMode(QtWidgets.QLineEdit.Normal)

def mes\_box(self, text):

self.messagebox = QMessageBox(self)

self.messagebox.setWindowTitle("Ошибка")

self.messagebox.setText(text)

self.messagebox.setStandardButtons(QMessageBox.Ok)

self.messagebox.show()

def enter(self):

auth\_log = self.ui.edit\_login.text()

auth\_pas = self.ui.edit\_password.text()

if auth\_log == 'admin' and auth\_pas == 'admin':

self.parent().page\_id = [0, 1, 2, 3, 4, 5]

self.parent().show()

self.close()

if auth\_log == '' or auth\_pas == '':

self.mes\_box('Заполните все поля!')

logging.log(logging.INFO, 'Ошибка!')

else:

self.parent().id, password, access = self.parent().db.get\_pas(auth\_log)

if password != auth\_pas:

self.mes\_box('Неверный логин или пароль')

logging.log(logging.INFO, 'Ошибка!')

elif password == auth\_pas:

if access == 0:

self.parent().page\_id = [0, 2, 4, 5]

self.parent().ui.stackedWidget\_main.setCurrentIndex(self.parent().page\_id[0])

self.parent().show()

self.close()

elif access == 1:

self.parent().page\_id = [1, 3]

self.parent().ui.stackedWidget\_main.setCurrentIndex(self.parent().page\_id[0])

self.parent().show()

self.close()

elif access == 2:

self.mes\_box('Отказано в доступе')

logging.log(logging.INFO, 'Ошибка!')

class Builder:

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

self.qapp = QApplication(sys.argv)

self.window = MainWindow()

self.auth()

def auth(self):

self.window.open\_auth()

self.qapp.exec()

logging.log(logging.INFO, 'Приложение завершила свою работу.')

if \_\_name\_\_ == '\_\_main\_\_':

B = Builder()

**Б.2 Листинг кода “database.py”**

import sqlite3

class DataBase:

def \_\_init\_\_(self, name='data.db'):

self.db = sqlite3.connect(f"{name}")

cur = self.db.cursor()

"""

Блок инициализация таблиц

"""

cur.execute("""CREATE TABLE IF NOT EXISTS Readers (

Reader\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Reader\_Name TEXT,

Reader\_Date DATE,

Reader\_Address TEXT,

Reader\_Phone TEXT

)

""")

cur.execute("""CREATE TABLE IF NOT EXISTS Publishers (

Publisher\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Publisher\_Name TEXT,

Publisher\_Address TEXT,

Publisher\_Phone TEXT,

Publisher\_Site TEXT

)

""")

cur.execute("""CREATE TABLE IF NOT EXISTS Books (

Book\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Book\_Name TEXT,

Book\_Author TEXT,

Book\_Date DATE,

Publisher\_ID INT,

FOREIGN KEY (Publisher\_ID) REFERENCES Publishers(Publisher\_ID) ON DELETE CASCADE ON UPDATE CASCADE

)

""")

cur.execute("""CREATE TABLE IF NOT EXISTS Employees (

Employee\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Employee\_Name TEXT,

Employee\_Date DATE,

Employee\_Address TEXT,

Employee\_Passport TEXT,

Employee\_Phone TEXT,

Login TEXT,

Password TEXT,

Access\_Level INT,

Position\_ID INT,

FOREIGN KEY (Position\_ID) REFERENCES Positions(Position\_ID) ON DELETE CASCADE ON UPDATE CASCADE

)

""")

cur.execute("""CREATE TABLE IF NOT EXISTS Positions (

Position\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Position\_Name TEXT,

Employee\_Salary INT

)

""")

cur.execute("""CREATE TABLE IF NOT EXISTS Issues (

Issue\_ID INTEGER PRIMARY KEY AUTOINCREMENT,

Issue\_Date DATE,

Return\_Status DATE,

Reader\_ID INT,

Book\_ID INT,

Employee\_ID INT,

FOREIGN KEY (Reader\_ID) REFERENCES Readers(Reader\_ID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (Book\_ID) REFERENCES Books(Book\_ID) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (Employee\_ID) REFERENCES Employees(Employee\_ID) ON DELETE CASCADE ON UPDATE CASCADE

)

""")

self.db.commit()

cur.close()

"""

Блок чтение данных из таблиц

"""

def get\_from\_books(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Books""")

records = cur.fetchall()

cur.close()

return records

def get\_from\_readers(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Readers""")

records = cur.fetchall()

cur.close()

return records

def get\_from\_issues(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Issues""")

records = cur.fetchall()

cur.close()

return records

def get\_from\_publishers(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Publishers""")

records = cur.fetchall()

cur.close()

return records

def get\_from\_positions(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Positions""")

records = cur.fetchall()

cur.close()

return records

def get\_from\_employees(self):

cur = self.db.cursor()

cur.execute("""SELECT \* FROM Employees""")

records = cur.fetchall()

cur.close()

return records

def get\_pas(self, log):

cur = self.db.cursor()

try:

cur.execute(f"""SELECT Employee\_ID, Password, Access\_Level FROM Employees WHERE Login='{log}'""")

rec = cur.fetchall()[0]

cur.close()

return rec[0], rec[1], rec[2]

except Exception:

cur.close()

return '', '', ''

"""

Блок создания комбобоксов

"""

def create\_combobox\_positions(self): # Данные для комбобокса Positions

cur = self.db.cursor()

cur.execute("""SELECT Position\_ID, Position\_Name FROM Positions""")

records = cur.fetchall()

l = []

for i in records:

l.append(str(i[0]) + ' ' + i[1])

cur.close()

return l

def create\_combobox\_employees(self): # Данные для комбобокса Employees

cur = self.db.cursor()

cur.execute(f"""SELECT Employee\_ID, Employee\_Name FROM Employees WHERE Access\_Level = 0""")

records = cur.fetchall()

l = []

for i in records:

l.append(str(i[0]) + ' ' + i[1])

cur.close()

return l

def create\_combobox\_readers(self): # Данные для комбобокса Readers

cur = self.db.cursor()

cur.execute("""SELECT Reader\_ID, Reader\_Name FROM Readers""")

records = cur.fetchall()

l = []

for i in records:

l.append(str(i[0]) + ' ' + i[1])

cur.close()

return l

def create\_combobox\_books(self): # Данные для комбобокса Books

cur = self.db.cursor()

cur.execute("""SELECT Book\_ID, Book\_Name FROM Books""")

records = cur.fetchall()

l = []

for i in records:

l.append(str(i[0]) + ' ' + i[1])

cur.close()

return l

def create\_combobox\_publishers(self): # Данные для комбобокса Publishers

cur = self.db.cursor()

cur.execute(f"""SELECT Publisher\_ID, Publisher\_Name FROM Publishers""")

records = cur.fetchall()

l = []

for i in records:

l.append(str(i[0]) + ' ' + i[1])

cur.close()

return l

"""

Блок добавления данных в таблицы

"""

def add\_in\_books(self, name, author, date, publisher\_id):

cur = self.db.cursor()

cur.execute("INSERT INTO Books VALUES (NULL, ?, ?, ?, ?)", (name, author, date, publisher\_id))

self.db.commit()

cur.close()

def add\_in\_positions(self, name, salary):

cur = self.db.cursor()

cur.execute("INSERT INTO Positions VALUES (NULL, ?, ?)", (name, salary))

self.db.commit()

cur.close()

def add\_in\_issues(self, is\_date, status, reader\_id, book\_id, employee\_id):

cur = self.db.cursor()

cur.execute("INSERT INTO Issues VALUES (NULL, ?, ?, ?, ?, ?)", (is\_date, status, reader\_id, book\_id, employee\_id))

self.db.commit()

cur.close()

def add\_in\_employees(self, name, date, address, passport, phone, login, password, access, position\_id):

cur = self.db.cursor()

cur.execute("INSERT INTO Employees VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (name, date, address, passport, phone, login, password, access, position\_id))

self.db.commit()

cur.close()

def add\_in\_readers(self, name, date, address, phone):

cur = self.db.cursor()

cur.execute("INSERT INTO Readers VALUES (NULL, ?, ?, ?, ?)", (name, date, address, phone))

self.db.commit()

cur.close()

def add\_in\_publishers(self, name, address, phone, site):

cur = self.db.cursor()

cur.execute("INSERT INTO Publishers VALUES (NULL, ?, ?, ?, ?)", (name, address, phone, site))

self.db.commit()

cur.close()

"""

Блок удаления данных

"""

def delete\_from\_books(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Books WHERE Book\_ID={id}""")

self.db.commit()

cur.close()

def delete\_from\_issues(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Issues WHERE Issue\_ID={id}""")

self.db.commit()

cur.close()

def delete\_from\_readers(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Readers WHERE Reader\_ID={id}""")

self.db.commit()

cur.close()

def delete\_from\_publishers(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Publishers WHERE Publisher\_ID={id}""")

self.db.commit()

cur.close()

def delete\_from\_employees(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Employees WHERE Employee\_ID={id}""")

self.db.commit()

cur.close()

def delete\_from\_positions(self, id):

id = int(id)

cur = self.db.cursor()

cur.execute(f"""DELETE from Positions WHERE Position\_ID={id}""")

self.db.commit()

cur.close()

"""

Блок обновления данных

"""

def update\_books(self, id, name, description, date, publisher\_id):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Books set Book\_Name="{name}", Book\_Author="{description}", Book\_Date="{date}", Publisher\_ID={publisher\_id} WHERE Book\_ID={id}""")

self.db.commit()

cur.close()

def update\_issues(self, id, is\_date, status, reader\_id, book\_id, employee\_id):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Issues set Issue\_Date="{is\_date}", Return\_Status="{status}", Reader\_ID={reader\_id}, Book\_ID={book\_id}, Employee\_ID={employee\_id} WHERE Issue\_ID={id}""")

self.db.commit()

cur.close()

def update\_positions(self, id, name, salary):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Positions set Position\_Name="{name}", Employee\_Salary={salary} WHERE Position\_ID={id}""")

self.db.commit()

cur.close()

def update\_readers(self, id, name, date, address, phone):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Readers set Reader\_Name="{name}", Reader\_Date="{date}", Reader\_Address="{address}", Reader\_Phone="{phone}" WHERE Reader\_ID={id}""")

self.db.commit()

cur.close()

def update\_publishers(self, id, name, address, phone, site):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Publishers set Publisher\_Name="{name}", Publisher\_Address="{address}", Publisher\_Phone="{phone}", Publisher\_Site="{site}" WHERE Publisher\_ID={id}""")

self.db.commit()

cur.close()

def update\_employees(self, id, name, date, address, passport, phone, login, password, access, position\_id):

id = int(id)

cur = self.db.cursor()

cur.execute(

f""" UPDATE Employees set Employee\_Name="{name}", Employee\_Date="{date}", Employee\_Address="{address}", Employee\_Passport="{passport}", Employee\_Phone="{phone}",

Login="{login}", Password="{password}", Access\_Level={access}, Position\_ID={position\_id} WHERE Employee\_ID={id}""")

self.db.commit()

cur.close()

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

db = DataBase()

# ПРИЛОЖЕННИЕ В. Тесты программных модулей

import sys

from unittest import TestCase

from PyQt5 import QtCore

from PyQt5.QtCore import QDate, QItemSelectionModel

from PyQt5.QtTest import QTest

from PyQt5.QtWidgets import QApplication

from database import DataBase

from gui import MainWindow

class TestAddData(TestCase):

def setUp(self):

self.qapp = QApplication(sys.argv)

self.db = DataBase()

self.window = MainWindow()

def test\_add\_a\_position\_lib(self):

btn\_add = self.window.ui.add\_pos # Объявление кнопки

self.window.ui.add\_pos\_name.setText("Библиотекарь")

self.window.ui.add\_pos\_salary.setValue(10000)

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_a\_position\_otd(self):

btn\_add = self.window.ui.add\_pos # Объявление кнопки

self.window.ui.add\_pos\_name.setText("Сотрудник отдела кадров")

self.window.ui.add\_pos\_salary.setValue(10000)

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_employee\_lib(self):

btn\_add = self.window.ui.add\_emloyee # Объявление кнопки

self.window.ui.employees\_fio.setText("Максим")

self.window.ui.employees\_date.setDate(QDate.fromString("24.01.2003"))

self.window.ui.employees\_address.setText("test")

self.window.ui.employees\_passport.setText("5456789098")

self.window.ui.employees\_phone.setText("9296781708")

self.window.ui.login.setText("test")

self.window.ui.password.setText("test")

self.window.ui.access.setValue(1)

self.window.ui.positions\_combobox.addItems(self.db.create\_combobox\_positions())

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_employee\_otd(self):

btn\_add = self.window.ui.add\_emloyee # Объявление кнопки

self.window.ui.employees\_fio.setText("Александр")

self.window.ui.employees\_date.setDate(QDate.fromString("04.01.2003"))

self.window.ui.employees\_address.setText("test")

self.window.ui.employees\_passport.setText("9856789098")

self.window.ui.employees\_phone.setText("9296971708")

self.window.ui.login.setText("test2")

self.window.ui.password.setText("test2")

self.window.ui.access.setValue(0)

self.window.ui.positions\_combobox.addItems(self.db.create\_combobox\_positions())

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_a\_publisher(self):

btn\_add = self.window.ui.pushButton\_2 # Объявление кнопки

self.window.ui.lineEdit\_2.setText("Издательство")

self.window.ui.lineEdit\_3.setText("test\_test")

self.window.ui.lineEdit\_4.setText("9296971708")

self.window.ui.lineEdit\_5.setText("test.ru")

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_book(self):

btn\_add = self.window.ui.add\_dep # Объявление кнопки

self.window.ui.dep\_name\_line.setText("Война и Мир")

self.window.ui.percent\_line.setText("Л.Толстой")

self.window.ui.dateEdit.setDate(QDate.fromString("2000"))

self.window.ui.publishers\_combobox.addItems(self.db.create\_combobox\_publishers())

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_a\_reader(self):

btn\_add = self.window.ui.emp\_add # Объявление кнопки

self.window.ui.add\_fio\_reader.setText("Пётр")

self.window.ui.add\_date\_birth\_reader.setDate(QDate.fromString("01.01.2022"))

self.window.ui.add\_address\_reader.setText("test\_test")

self.window.ui.add\_phone\_reader.setText("9296971708")

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_issue(self):

btn\_add = self.window.ui.issues\_add # Объявление кнопки

self.window.ui.issues\_date.setDate(QDate.fromString("01.01.2022"))

self.window.ui.issues\_status.setText("01.01.2023")

self.window.ui.readers\_combobox.addItems(self.db.create\_combobox\_readers())

self.window.ui.books\_combobox.addItems(self.db.create\_combobox\_books())

self.window.ui.employee\_combobox.addItems(self.db.create\_combobox\_employees())

QTest.mouseClick(btn\_add, QtCore.Qt.MouseButton.LeftButton)

self.db.delete\_from\_issues(1)

self.db.delete\_from\_books(1)

self.db.delete\_from\_readers(1)

self.db.delete\_from\_employees(1)

self.db.delete\_from\_positions(1)

self.db.delete\_from\_publishers(1)

class TestDeleteAndSave(TestCase):

def setUp(self):

self.qapp = QApplication(sys.argv)

self.db = DataBase()

self.window = MainWindow()

def test\_add\_save\_position(self):

self.db.add\_in\_positions("name", 20000)

QTest.mouseClick(self.window.ui.save\_pos, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_save\_publisher(self):

self.db.add\_in\_publishers("name", "address", "phone", "site")

QTest.mouseClick(self.window.ui.pushButton, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_save\_employee(self):

self.db.add\_in\_employees("name", "date", "address", "passport", "phone", "test1", "test1", 0, 1)

QTest.mouseClick(self.window.ui.save\_emloyee, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_save\_book(self):

self.db.add\_in\_books("name", "author", "date", 1)

QTest.mouseClick(self.window.ui.save\_dep, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_save\_reader(self):

self.db.add\_in\_readers("name", "date", "address", "phone")

QTest.mouseClick(self.window.ui.emp\_save, QtCore.Qt.MouseButton.LeftButton)

def test\_add\_save\_issue(self):

self.db.add\_in\_issues("date", "status", 1, 1, 1)

QTest.mouseClick(self.window.ui.issues\_save, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_a\_issue(self):

rowcount = self.window.table\_issues.rowCount()

self.window.table\_issues.setCurrentCell(rowcount-1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.issues\_delete

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.issues\_save, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_book(self):

rowcount = self.window.table\_books.rowCount()

self.window.table\_books.setCurrentCell(rowcount-1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.delete\_dep

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.save\_dep, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_employee(self):

rowcount = self.window.table\_employees.rowCount()

self.window.table\_employees.setCurrentCell(rowcount-1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.delete\_emloyee

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.save\_emloyee, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_position(self):

rowcount = self.window.table\_positions.rowCount()

self.window.table\_positions.setCurrentCell(rowcount - 1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.delete\_pos

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.save\_pos, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_publisher(self):

rowcount = self.window.table\_publishers.rowCount()

self.window.table\_publishers.setCurrentCell(rowcount - 1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.pushButton\_3

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.pushButton, QtCore.Qt.MouseButton.LeftButton)

def test\_delete\_reader(self):

rowcount = self.window.table\_readers.rowCount()

self.window.table\_readers.setCurrentCell(rowcount-1, 1, QItemSelectionModel.SelectionFlag.Select)

btn\_del = self.window.ui.emp\_delete

QTest.mouseClick(btn\_del, QtCore.Qt.MouseButton.LeftButton)

QTest.mouseClick(self.window.ui.emp\_save, QtCore.Qt.MouseButton.LeftButton)