

Колледж космического машиностроения и технологий

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Приложение "Управление заказами"»**

Выполнил студент

Батраков Д.С.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королев 2021 г.**

Содержание

[Введение 3](#_Toc75716081)

[1 Теоретическая часть 4](#_Toc75716082)

[1.1 Описание предметной области 4](#_Toc75716083)

[1.2 Описание существующих разработок 5](#_Toc75716084)

[2 Проектная часть 8](#_Toc75716085)

[1.3 Диаграмма прецедентов 8](#_Toc75716086)

[1.4 Выбор инструментов 8](#_Toc75716087)

[1.4.1 Язык программирования 8](#_Toc75716088)

[1.4.2 База данных 10](#_Toc75716089)

[1.5 Проектирование сценария 10](#_Toc75716090)

[1.6 Диаграммы классов 11](#_Toc75716091)

[1.7 Описание главного модуля. 11](#_Toc75716092)

[1.8 Описание спецификаций к модулям 12](#_Toc75716093)

[1.9 Описание модулей 13](#_Toc75716094)

[1.10 Описание применения средств отладки 17](#_Toc75716095)

[2 Эксплуатационная часть 19](#_Toc75716096)

[2.1 Руководство оператора 19](#_Toc75716097)

[2.1.1 Назначение программы 19](#_Toc75716098)

[2.1.2 Условия выполнения программы 19](#_Toc75716099)

[2.1.3 Выполнение программы 19](#_Toc75716100)

[2.1.4 Сообщение оператору 21](#_Toc75716101)

[2.2 To-Do лист 21](#_Toc75716102)

[Заключение 22](#_Toc75716103)

[Список литературы и интернет ресурсов 23](#_Toc75716104)

[Приложения 24](#_Toc75716105)

# Введение

Данный курсовой проект заключается в разработке приложения, для быстрого и удобного взаимодействия с базами данных. Разработка приложения будет происходить на Python.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# Теоретическая часть

## Описание предметной области

**Управление заказами** – объемное понятие, которое подразумевает контроль на каждом этапе цикла: от момента оформления заявки до доставки товара покупателю.

**Быстрая и корректная обработка заказов** – одно из важнейших конкурентных преимуществ в электронной коммерции. Оформляя покупку в интернет-магазине, потребитель ожидает получить товар в кратчайшие сроки. И он не придет на сайт повторно, если не будет мгновенной обратной связи, возникнут какие-либо накладки или заказ попросту потеряется среди десятков других. Чтобы исключить подобные ситуации, **нужно обеспечить грамотное и эффективное управление заказами покупателей**.

**Главная задача управления – качественная и четкая отработка всех этапов.** Этот процесс является одним из элементов общей концепции обслуживания покупателей. При условии правильной организации он помогает увеличить лояльность аудитории и поддерживать положительную репутацию компании.

**Эффективность управления зависит от следующих факторов:**

* **систематизация**. Информацию о товарах и текущих статусах сделок желательно структурировать и хранить в единой системе;
* **прозрачность**. Важно своевременно фиксировать и передавать актуальные данные, иметь наглядное представление цепочки поставок, оперативно выявлять и устранять проблемы;
* **аналитика**. Необходимо прогнозировать и анализировать важные для компании метрики, настраивать процессы, исходя из намеченных показателей. Прогнозирование объемов продаж позволяет создать достаточное количество запасов на складе, что способствует быстрому удовлетворению запросов клиента;
* **гибкость**. Следует разделять заказы на группы и сегменты, ставить отдельные задачи и передавать их в соответствующие отделы и системы.

## Описание существующих разработок

На данный момент существует множество уже разработанных приложений по этой тематике. Практически все из них предоставляют похожий функционал:

* размещение заказа в базе;
* проверка данных;
* сверка с актуальными запасами на складе;
* подтверждение покупки;
* создание счетов;
* прием оплаты;
* отгрузка и упаковка;
* передача груза перевозчику;
* информирование покупателя о текущем статусе заявки;
* доставка;
* обновление информации о доступных остатках в товарных карточках.

Примеры сайтов:

* RetailCRM[1]

помогает продавать товары в интернете, работать с клиентской базой, автоматизировать обработку заказов, анализировать статистику, вести email-маркетинг.

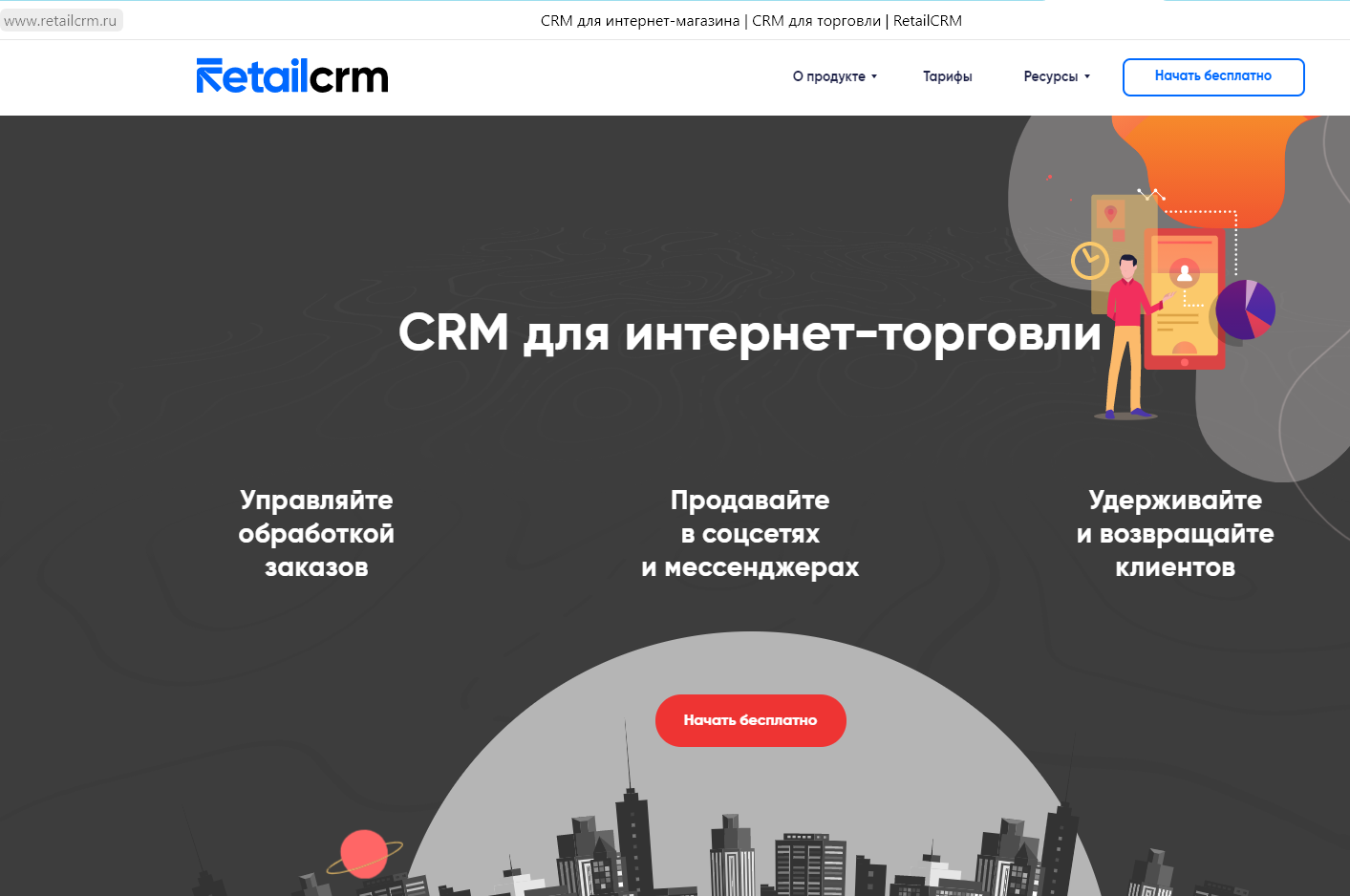


Рисунок 1. Окно сайта RetailCRM

* Ekam [2]

Возможность загружать товары из Excel или создавать в самой системе; Создавать оприходование или приемку товара; Управление заказами из вашего интернет-магазина; Отслеживайте остатки: фактические и с учетом резерва; Формирование отчётов по дням, сотрудникам, проданным позициям, контроль работы персонала.

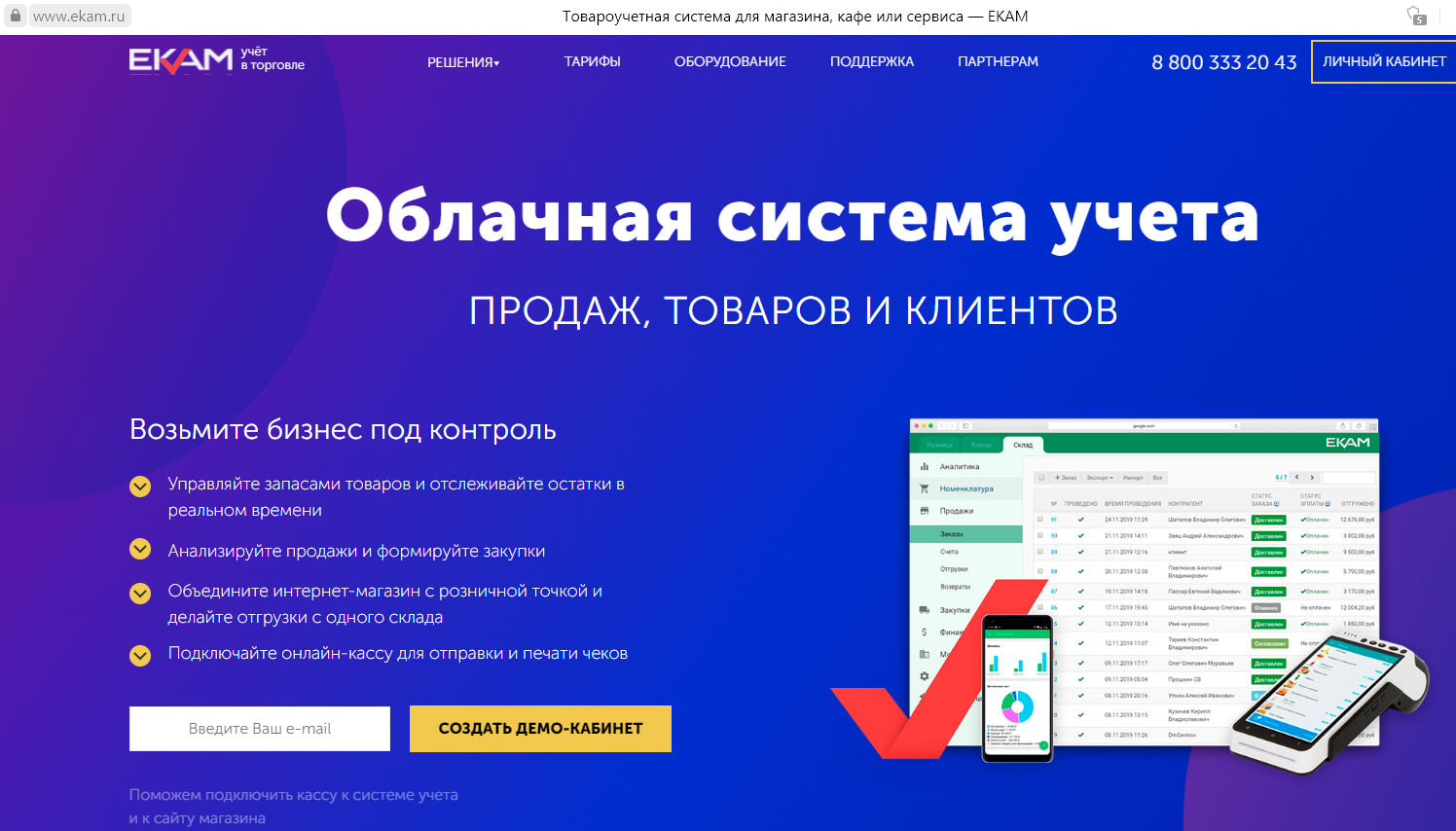


Рисунок 2. Окно сайта Ekam

* generixgroup [3]

визуальный доступ к запасам и остаткам для специалистов, помощь в учете и аналитике;

эффективное планирование и контроль логистических операций;

поддержку продажи, позволяющую в точности выполнить обязательства перед клиентом;

оптимизацию бизнес-процессов и документооборота, связанного с закрытием заявок;

снижение логистических издержек.

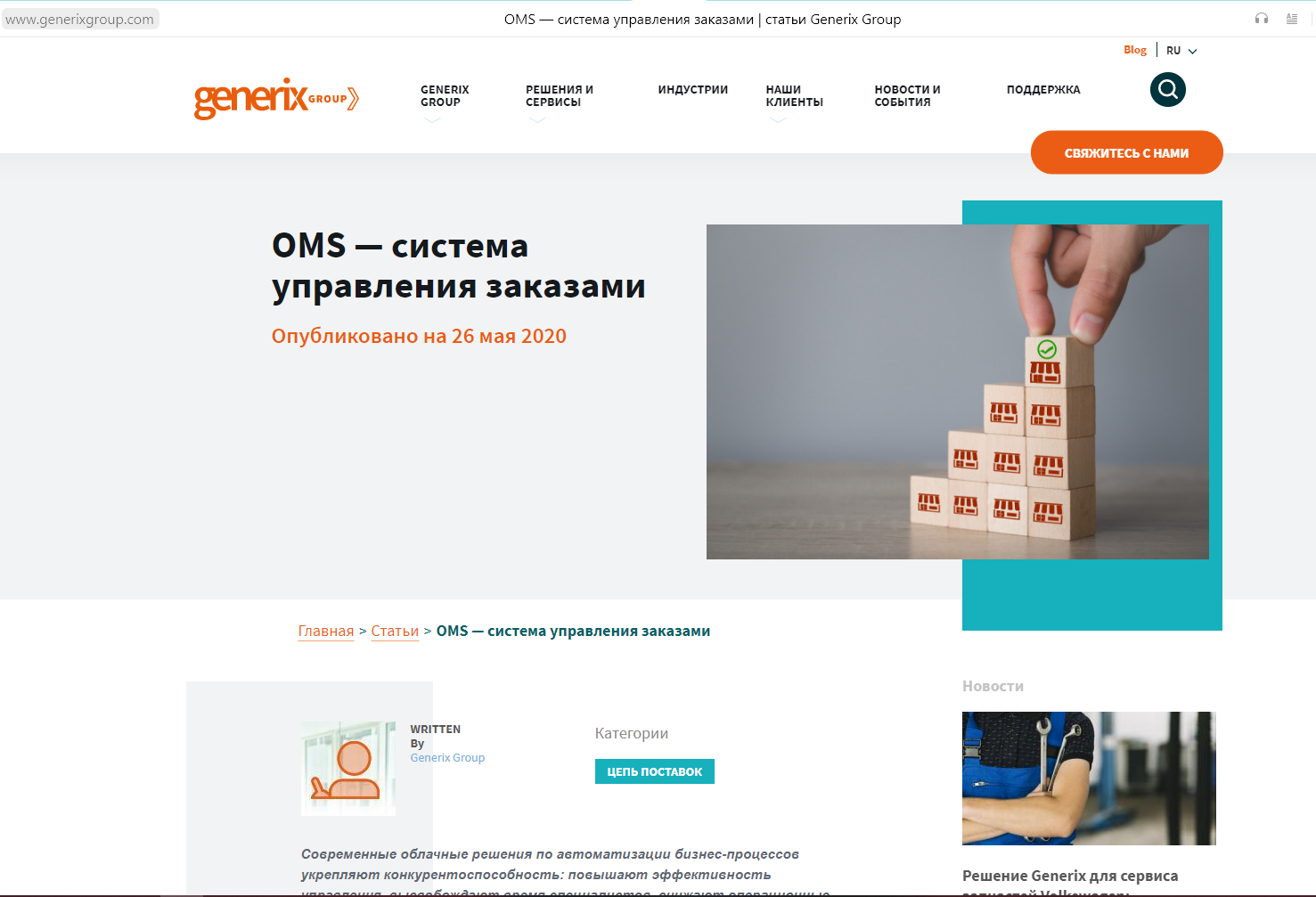


Рисунок 3. Окно сайта generixgroup

# 2 Проектная часть

## Диаграмма прецедентов

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения.

Администратор

Рисунок 4. Диаграмма прецедентов приложения.

## Выбор инструментов

В разработке приложения “Управление заказами”, используется Python, в качестве СУБД использовался SQLite и в качестве IDE используется PyCharm и компилятор GCC, который встроен в PyCharm.

### Язык программирования

**Python** - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным - всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скрипто. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектноориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается через декораторы, более полноценная поддержка обеспечивается дополнительными фреймворками. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений.

Стандартная библиотека включает большой набор полезных переносимых функций, начиная от функционала для работы с текстом и заканчивая средствами для написания сетевых приложений. Дополнительные возможности, такие как математическое моделирование, работа с оборудованием, написание веб-приложений или разработка игр, могут реализовываться посредством обширного количества сторонних библиотек, а также интеграцией библиотек, написанных на Си или C++, при этом и сам интерпретатор Python может интегрироваться в проекты, написанные на этих языках. Существует и специализированный репозиторий программного обеспечения, написанного на Python, — PyPI. Данный репозиторий предоставляет средства для простой установки пакетов в операционную систему и стал стандартом де-факто для Python. По состоянию на 2019 год в нём содержалось более 175 тысяч пакетов.

### База данных

В разработке приложений под Ubuntu в основном в качестве базы данных используется **SQLite**. Причины использования:

1. **Минимальные затраты ресурсов.** Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера: база данных SQLite представляет собой обычный файл. Когда БД не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.
2. **Оптимизация для одного пользователя.** С базой данных взаимодействует только наше приложение. Поэтому можно обойтись без идентификации с именем пользователя и паролем.
3. **Надежность и быстрота.** Базы данных SQLite поддерживают транзакции баз данных. Кроме того, операции чтения и записи данных реализуются на оптимизированном коде С.

## Проектирование сценария

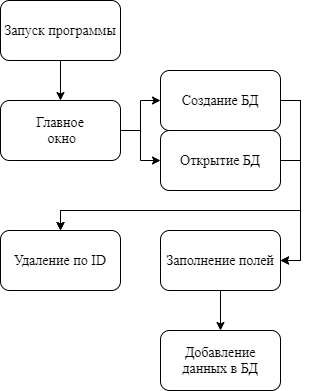
В данном разделе приведен сценарий использования приложения пользователем (Рисунок 5).

Рисунок 5. Сценарий использования сайта.

При запуске приложения открывается главный экран для ввода данных. Для работы с программой необходимо открыть или создать новый файл типа «Data Base File», сделать это можно на панели функций. После ввода данных и нажатия кнопки «Добавить» все введенные данные сохраняются в ранее открытую БД.

## C:\Users\User\Desktop\Курсач\Диаграммы\Диаграма классов.pngДиаграммы классов

В данном разделе располагается диаграмма классов сайта. (Рисунок 6).

Рисунок 6. Диаграмма классов приложения

## Описание главного модуля.

Главный модуль состоит из класса Database. Он выполняется при запуске приложения. Данный модуль подключается к базе данных и создаёт основную таблицу и функции.

Листинг 1. Главный модуль

from tkinter import \*

from tkinter import messagebox

from tkinter import filedialog

import sqlite3

TEXT\_BUTTON\_DEL = "Удалить"

TEXT\_CREATE\_FILE = "Создать базу данных"

TEXT\_OPEN\_FILE = "Открыть базу данных"

nameDB = "database.db"

BUTTON\_COLOR = "#E6E6FA"

WINDOW\_COLOR = "#FF8C00"

ACT\_BUTTON\_COLOR = "#B0C4DE"

LABEL\_BG = "#FF8C00"

class Database:

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

# подключение к базе данных

self.db = sqlite3.connect(nameDB)

self.cursor = self.db.cursor()

def create\_table(self, table):

# создание таблицы со столбцами в базе данных

self.cursor.execute(f"CREATE TABLE IF NOT EXISTS '{table}' (\

order\_id INTEGER, \

order\_name TEXT, \

customer\_name TEXT, \

number INTEGER, \

price INTEGER)")

## Описание спецификаций к модулям

Всего разработано 3 модуля:

* Инициализация базы данных (Database)
* Утилитарные функции (Main)
* Работа с окном приложения (Window)

Взаимодействие между модулями можно наглядно увидеть по диаграммам классов из раздела 2.4.

## Описание модулей

**Инициализация базы данных** – модуль, который содержит класс главной формы программы.

Листинг 2. Инициализация базы данных

class Database:

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

# подключение к базе данных

self.db = sqlite3.connect(nameDB)

self.cursor = self.db.cursor()

def create\_table(self, table):

# создание таблицы со столбцами в базе данных

self.cursor.execute(f"CREATE TABLE IF NOT EXISTS '{table}' (\

order\_id INTEGER, \

order\_name TEXT, \

customer\_name TEXT, \

number INTEGER, \

price INTEGER)")

def save(self):

# сохранение созданной таблицы базы данных

self.db.commit()

def get\_table(self, table):

# вызов таблицы базы данных

return [i for i in self.cursor.execute(f"SELECT \* FROM {table}")]

**Утилитарные функции** – содержит утилитарные функции по типу: создание новой или открытия уже существующей базы данных, добавление данных в таблицу и удаление строки из базы данных с помощью его id.

Листинг 3. Утилитарные функции

class Main:

def delete\_order\_by\_id():

# удаление строки из базы данных с помощью его id через окно приложения

db = sqlite3.connect(nameDB)

cursor = db.cursor()

db.execute('DELETE FROM {0} WHERE order\_id = {1}'.format("Orders", window.id\_order\_del.get()))

db.commit()

def insert\_data():

# добавление данных в таблицу через окно приложения

db = Database(nameDB)

db.create\_table("Orders")

db.cursor.execute("INSERT INTO orders (order\_id, order\_name, customer\_name, number, price) VALUES (?,?,?,?,?)",

(window.id\_order.get(), window.name\_order.get(), window.customer.get(), window.number.get(), window.price.get()))

db.save()

def creat\_db():

# диалоговое окно, где в поле имя файлов надо ввести имя файла которое вы хотите создать

global nameDB

path\_db = filedialog.asksaveasfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*")))

db = sqlite3.connect(path\_db)

nameDB = path\_db

def open\_db():

# диалоговое окно, где пользователь выбирает какой файл ему нужно открыть

global nameDB

path\_db = filedialog.askopenfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*")))

# op является путём и его надо передать в функции добавления и удаления из файла

db = sqlite3.connect(path\_db)

nameDB = path\_db

**Работа с окном приложения** – присваивает строку из виджета поля для ввода текста, принимает строку и координаты, отображает текст.

Листинг 4. Работа с окном приложения

class Window:

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

# присваивает строку из виджета поля для ввода текста

self.window = Tk()

self.window.title("Управление заказами")

self.window.geometry("640x250")

self.window["bg"] = WINDOW\_COLOR

self.id\_order = StringVar()

self.name\_order = StringVar()

self.customer = StringVar()

self.number = StringVar()

self.price = StringVar()

self.id\_order\_del = StringVar()

self.top\_menu()

self.main\_menu()

def top\_menu(self):

# иницилизирую объект меню для секций

self.menu = Menu(self.window)

self.menu.add\_cascade(label=TEXT\_CREATE\_FILE, command=creat\_db)

self.menu.add\_cascade(label=TEXT\_OPEN\_FILE, command=open\_db)

self.menu.add\_cascade(label="О программе", command=lambda: messagebox.showinfo("О программе", "Разработку ведет Батраков Данила <3. ")) # Меню о программе добавляется на панель меню

self.window.config(menu=self.menu) # доступ к атрибуту объекта после его инициализации

def create\_label(self, text\_label, x\_label, y\_label):

# Функция принимает строку и координаты

Label(text=text\_label, bg=LABEL\_BG).place(x=x\_label, y=y\_label) # Отображает текст в окне по переданным координатам

def main\_menu(self):

# отображаем текст хранящейся в переменной

self.create\_label("ID заказа: ", 0, 3)

Entry(width=20, textvariable=self.id\_order).place(x=0, y=20)

# определим элемент Entry который представляет собой поле для ввода текста

self.create\_label("Заказ: ", 0, 40)

Entry(width=20, textvariable=self.name\_order).place(x=0, y=60)

self.create\_label("Заказчик:", 160, 3)

Entry(width=20, textvariable=self.customer).place(x=160, y=20)

self.create\_label("Номер телефона:", 160, 40)

Entry(width=20, textvariable=self.number).place(x=160, y=60)

self.create\_label("Цена: ", 320, 3)

Entry(width=20, textvariable=self.price).place(x=320, y=20)

Button(self.window, text="Добавить", command=insert\_data, bg=BUTTON\_COLOR,

activebackground=ACT\_BUTTON\_COLOR).place(x=382, y=45)

self.create\_label("Удаление по ID: ", 480, 3)

Entry(width=20, textvariable=self.id\_order\_del, ).place(x=480, y=20)

Button(self.window, text=TEXT\_BUTTON\_DEL, width= 8, command=delete\_order\_by\_id, bg=BUTTON\_COLOR,

activebackground=ACT\_BUTTON\_COLOR).place(x=540, y=45)

## Описание применения средств отладки

В ходе написания курсового проекта был найден баг. А именно при запуске программы не было функциональной кнопки «Удалить» (Рисунок 7):

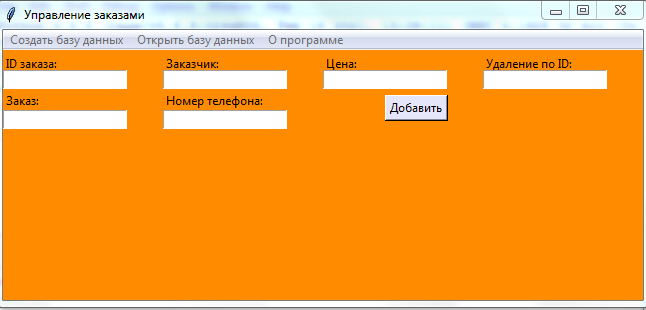


Рисунок 7. Ошибка

При проверке кода были исправлены найденные ошибки, в результате при запуске программы ошибок не было (Рисунок 8):

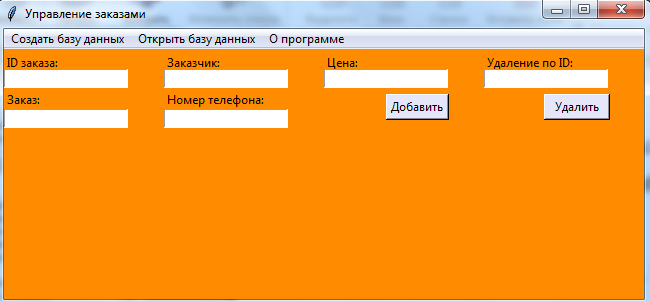


Рисунок 8. Успешная выдача результата.

# Эксплуатационная часть

## Руководство оператора

### Назначение программы

Разработанное приложение позволяет с легкостью создавать и заполнять базы данных для учета заказов. Каждая база данных сохраняется на устройстве пользователя со всеми введенными в неё данными.

### Условия выполнения программы

Любая ОС с двумя и более гигабайтами ОЗУ:

Windows XP, Windows Vista, Windows 7, Windows , Windows 10 , Linux, Apple Mac OS X .

### Выполнение программы

При запуске приложения вам откроется стартовое окно:

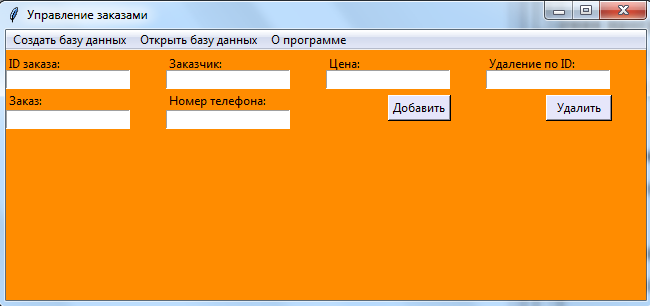


Рисунок 9. Главное окно

На главном окне необходимо создать новую или же открыть уже существующую базу данных, сделать это можно на функциональной панели сверху, соответствующими кнопками.



Рисунок 10. Функциональная панель

После того, как база данных будет открыта, вы можете приступать к заполнению пустых граф.

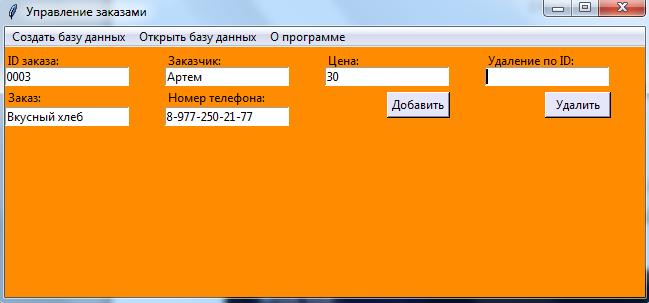


Рисунок 11. Заполненные графы

Для добавления данных в базу данных нажмите кнопку «Добавить», данных сохранятся в открытой базе данных



Рисунок 12. Кнопка завершения теста

После нажатия кнопки «Добавить» все введенные данные сохраняются в базе.

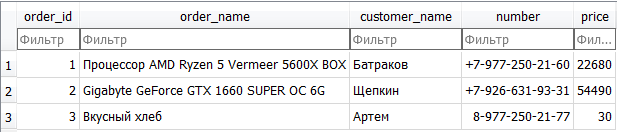


Рисунок 13. Сохраненные данные

### Сообщение оператору

Если база данных заранее не будет выбрана, данные в ней не сохранятся:

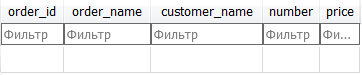


Рисунок 15. Результат добавления

**Причина:** не выбрана база данных.

**Действия программы:** программа не может сохранить данные в базе данных.

**Действия оператора:** создать/открыть баз данных.

## To-Do лист

* Добавить авторизацию
* Вывод ошибок при критических сбоях
* Просмотр базы данных в окне приложения

# Заключение

Благодаря данному приложению создавать базы данных для учета и управления заказами станет намного легче, так как не придётся тратить время на создание базы данных.

# Список литературы и интернет ресурсов

1. PyCharm

https://www.jetbrains.com/pycharm/

1. Документация Python

https://www.python.org/doc/

1. Работа с SQLite

https://zametkinapolyah.ru/zametki-o-mysql/tema-12-sql-zaprosy-select-v-bazax-dannyx-sqlite.html

1. SQLite

<https://devacademy.ru/article/sqlite-vs-mysql-vs-postgresql>

# Приложения

Приложение 1. Управление заказами.py

# Разработчики Андрей Слепов, Батраков Данила

from tkinter import \*

from tkinter import messagebox

from tkinter import filedialog

import sqlite3

TEXT\_BUTTON\_DEL = "Удалить"

TEXT\_CREATE\_FILE = "Создать базу данных"

TEXT\_OPEN\_FILE = "Открыть базу данных"

nameDB = "database.db"

BUTTON\_COLOR = "#E6E6FA"

WINDOW\_COLOR = "#FF8C00"

ACT\_BUTTON\_COLOR = "#B0C4DE"

LABEL\_BG = "#FF8C00"

class Database:

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

# подключение к базе данных

self.db = sqlite3.connect(nameDB)

self.cursor = self.db.cursor()

def create\_table(self, table):

# создание таблицы со столбцами в базе данных

self.cursor.execute(f"CREATE TABLE IF NOT EXISTS '{table}' (\

order\_id INTEGER, \

order\_name TEXT, \

customer\_name TEXT, \

number INTEGER, \

price INTEGER)")

def save(self):

# сохранение созданной таблицы базы данных

self.db.commit()

def get\_table(self, table):

# вызов таблицы базы данных

return [i for i in self.cursor.execute(f"SELECT \* FROM {table}")]

def delete\_order\_by\_id():

# удаление строки из базы данных с помощью его id через окно приложения

db = sqlite3.connect(nameDB)

cursor = db.cursor()

db.execute('DELETE FROM {0} WHERE order\_id = {1}'.format("Orders", window.id\_order\_del.get()))

db.commit()

def insert\_data():

# добавление данных в таблицу через окно приложения

db = Database(nameDB)

db.create\_table("Orders")

db.cursor.execute("INSERT INTO orders (order\_id, order\_name, customer\_name, number, price) VALUES (?,?,?,?,?)",

(window.id\_order.get(), window.name\_order.get(), window.customer.get(), window.number.get(), window.price.get()))

db.save()

def creat\_db():

# диалоговое окно, где в поле имя файлов надо ввести имя файла которое вы хотите создать

global nameDB

path\_db = filedialog.asksaveasfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*")))

db = sqlite3.connect(path\_db)

nameDB = path\_db

def open\_db():

# диалоговое окно, где пользователь выбирает какой файл ему нужно открыть

global nameDB

path\_db = filedialog.askopenfilename(initialdir="/", title="Select file", filetypes=(("database files", "\*.db"), ("all files", "\*.\*")))

# op является путём и его надо передать в функции добавления и удаления из файла

db = sqlite3.connect(path\_db)

nameDB = path\_db

class Window:

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

# присваивает строку из виджета поля для ввода текста

self.window = Tk()

self.window.title("Управление заказами")

self.window.geometry("640x250")

self.window["bg"] = WINDOW\_COLOR

self.id\_order = StringVar()

self.name\_order = StringVar()

self.customer = StringVar()

self.number = StringVar()

self.price = StringVar()

self.id\_order\_del = StringVar()

self.top\_menu()

self.main\_menu()

def top\_menu(self):

# иницилизирую объект меню для секций

self.menu = Menu(self.window)

self.menu.add\_cascade(label=TEXT\_CREATE\_FILE, command=creat\_db)

self.menu.add\_cascade(label=TEXT\_OPEN\_FILE, command=open\_db)

self.menu.add\_cascade(label="О программе", command=lambda: messagebox.showinfo("О программе", "Разработку ведет Батраков Данила <3. ")) # Меню о программе добавляется на панель меню

self.window.config(menu=self.menu) # доступ к атрибуту объекта после его инициализации

def create\_label(self, text\_label, x\_label, y\_label):

# Функция принимает строку и координаты

Label(text=text\_label, bg=LABEL\_BG).place(x=x\_label, y=y\_label) # Отображает текст в окне по переданным координатам

def main\_menu(self):

# отображаем текст хранящейся в переменной

self.create\_label("ID заказа: ", 0, 3)

Entry(width=20, textvariable=self.id\_order).place(x=0, y=20)

# определим элемент Entry который представляет собой поле для ввода текста

self.create\_label("Заказ: ", 0, 40)

Entry(width=20, textvariable=self.name\_order).place(x=0, y=60)

self.create\_label("Заказчик:", 160, 3)

Entry(width=20, textvariable=self.customer).place(x=160, y=20)

self.create\_label("Номер телефона:", 160, 40)

Entry(width=20, textvariable=self.number).place(x=160, y=60)

self.create\_label("Цена: ", 320, 3)

Entry(width=20, textvariable=self.price).place(x=320, y=20)

Button(self.window, text="Добавить", command=insert\_data, bg=BUTTON\_COLOR,

activebackground=ACT\_BUTTON\_COLOR).place(x=382, y=45)

self.create\_label("Удаление по ID: ", 480, 3)

Entry(width=20, textvariable=self.id\_order\_del, ).place(x=480, y=20)

Button(self.window, text=TEXT\_BUTTON\_DEL, width= 8, command=delete\_order\_by\_id, bg=BUTTON\_COLOR,

activebackground=ACT\_BUTTON\_COLOR).place(x=540, y=45)

window = Window()

mainloop() # отображает главное окно со всеми виджетами