

Санкт-Петербургский Национальный
Исследовательский Университет
Информационных технологий, механики и оптики

Домашняя работа

Реализация программной модели инфокоммуникационной системы

Выполнил: Шкода
Глеб Ярославович
Группа № К3123
Проверила: Казанова
Полина Петровна

Санкт-Петербург
2022

Цель работы:

Создать программное обеспечение учета грузового транспорта для Автотранспортного отдела логистической компании.

Задачи:

В программе должно быть реализовано:

1. Добавлять/удалять грузовой транспорт.
2. Просматривать весь доступный транспорт.
3. Просматривать грузовой транспорт по грузоподъёмности.
4. Просматривать свободный грузовой транспорт.
5. Вносить заявку на перевоз груза по указанным габаритам.
6. Подобрать и забронировать транспорт.
7. Просматривать занятый транспорт.
8. Интерфейс программы реализовать на ваше усмотрение.
9. Реализовать возможность сохранения данных в базу данных

Ход работы:

Отдельно от основной программы стоит реализовать объект, отвечающий за взаимодействие с базой данной sqlite3. Для этого создадим класс, при инициации которого будет создаваться новая база данных и таблица в ней. Эта таблица будет состоять из 7 столбцов: id машины, тип машины, её грузоподъёмность, максимальные длинна, ширина и высота перевозимого груза, а также информация о том, забронирована ли уже эта машина на перевозку. Отдельными методами реализуем, добавление новых элементов, удаление старых и прочее операции, которые пригодятся при работе основной части программы.

Небольшая часть реализации класса “База данных” (см. рисунок 1):

```
3 class DB:
4     def __init__(self):
5         self.con = sqlite3.connect('transport.db')
6         self.cursor = self.con.cursor()
7         command = 'CREATE TABLE IF NOT EXISTS Transport(ID INT, Тип TEXT, Грузоподъёмность REAL, Длина REAL, Ширина REAL, Высота REAL, Свободен INT);'
8         self.cursor.execute(command)
9     def insert_new(self, id, type, cap, l, w, h):
10        new_log = (id, type, cap, l, w, h, 1)
11        self.cursor.execute('INSERT INTO Transport VALUES(?, ?, ?, ?, ?, ?, ?);', new_log)
12        self.con.commit()
13    def delete(self, id):
14        self.cursor.execute('DELETE FROM Transport WHERE ID = {};'.format(id))
15        self.con.commit()
```

Рисунок 1. Создание базы данных, а также пример некоторых методов для неё.

Требуется создать программу с графическим интерфейсом, для этого будет использоваться модуль tkinter. Разберём, созданное с помощью этой библиотеке окно, которое приветствует пользователя при запуске программы (см. рисунок 2):

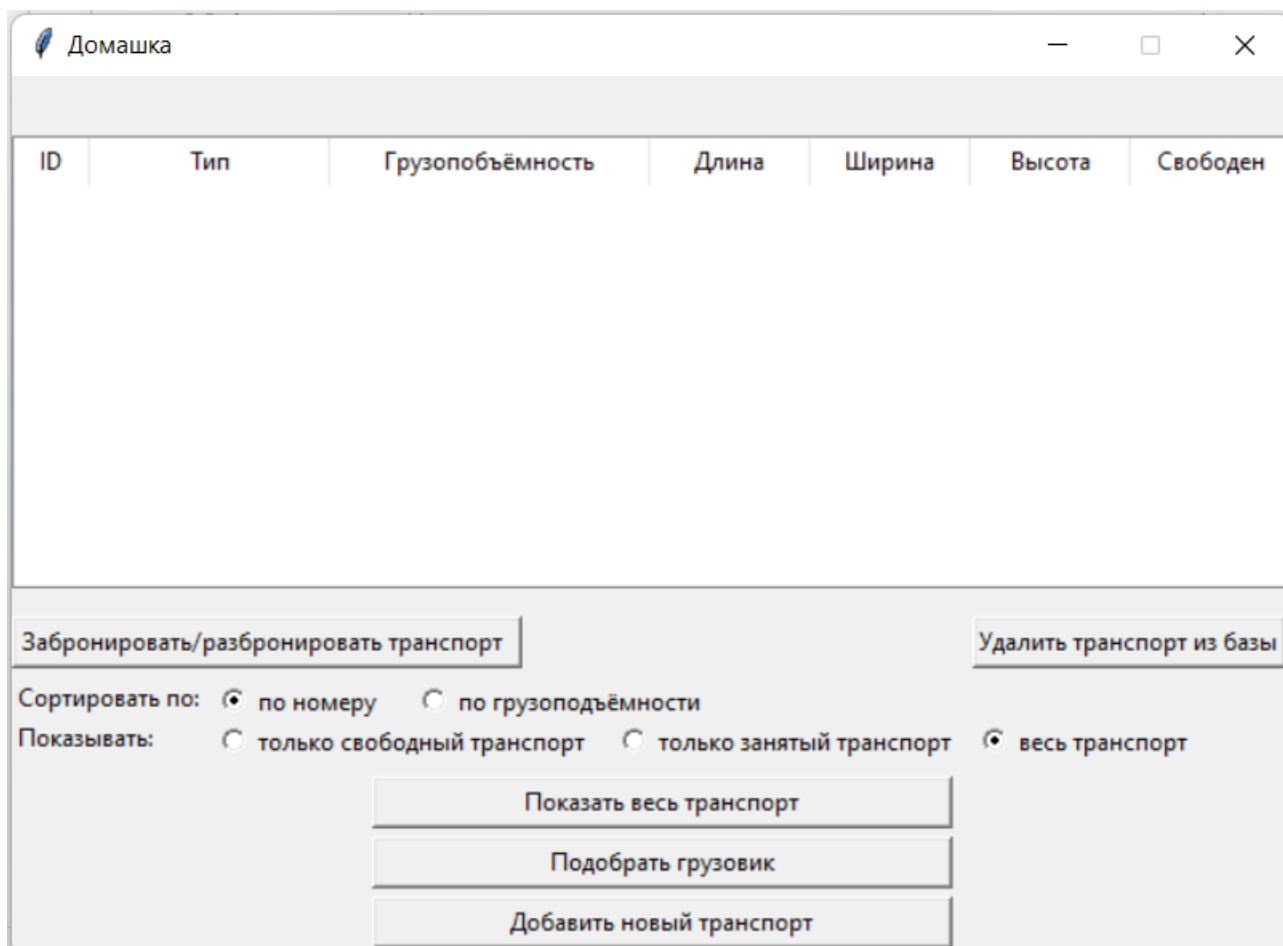


Рисунок 2. Стартовый вид окна программы.

Основным источником информации для пользователя будет виджет Treeview. Создадим с его помощью таблицу, состоящую из 7 заголовков. Таблицу реализуем как отдельный класс. На начальный момент времени, пользователем не было запрошено никакой информации о доступном транспорте поэтому таблица пуста. Ниже идут 2 кнопки: левая позволяет пользователю забронировать выбранный в таблице транспорт, или же отменить бронь, если он уже забронирован, при нажатии этой кнопки произойдет обновление записи о выбранном транспорте в базе данных. Правая кнопка позволяет удалить транспорт из базы данных, при нажатии выбранный транспорт пропадает из отображаемой таблицы, а также из базы данных. Далее предлагается настроить фильтр отображения транспорта, так машины можно отсортировать по их номеру в базе или по их грузоподъемности. Отображать можно весь доступный транспорт, а можно только свободный или, наоборот, только занятый. Кнопка ниже отображает в таблице весь доступный на данный момент транспорт. Следующая кнопка вызывает окно, для ввода пользователем информации о перевозимом грузе, после чего предлагаются только подходящие машины. Последняя кнопка вызывает вспомогательное окно, которое предлагает ввести информацию о новой машине, после чего информация о ней немедленно заносится в базу.

Рассмотрим каждый из этих виджетов подробнее.

Как уже было сказано, основная таблица – это отдельный класс созданный на основе виджета Treeview. При создании объекта определяются заголовки, после чего они выводятся на экран. Таблица также имеет методы полной очистки,

добавления новой строке, а также вывод элементов после соответствующей команды пользователя. Могут быть выведены как все элементы, так и подходящие под описанные пользователем фильтры. Также поддерживаются разные сортировки, в зависимости от предпочтений пользователя. Также поддерживается возможность удаление и бронирование выделенного пользователем элемента таблицы.

Например, метод бронирования реализован следующем образом (см. рисунок 3):

```
def book_unbook(self):
    q = self.table.focus()
    if q == '':
        messagebox.showerror(title='Ошибка', message='Ничего не выбрано')
        return
    q = self.table.item(q)
    q = q['values'].copy()
    for i in range(2, 6):
        q[i] = float(q[i])
    q_t = tuple(q)
    if q[-1] == 1:
        q[-1] = 0
        db.book(q[0])
    else:
        q[-1] = 1
        db.unbook(q[0])
    for i in range(len(self.all)):
        if self.all[i] == q_t:
            self.all[i] = tuple(q)
    self.update()
```

Рисунок 3. Метод бронирования/разбронирования машины.

Для бронирования необходимо, убедиться, что пользователь выбрал какой-либо транспорт, иначе сообщить ему об этом. После чего сменить статус транспорта в базе данных. Затем найти его в списке отображаемых элементов (он там гарантировано присутствует, ведь выбор сделан именно оттуда) и обновить информацию об элементе там. Затем нужно обновить отображаемую пользователю таблицу, так как данные там устарели. Удаление работает по схожему принципу, но вместо обновления информации о транспорте её достаточно просто удалить и обновить отображаемую таблицу.

Опции для сортировки реализованы с помощью виджета Radiobutton, к каждому ряду соответственно привязана переменная, которая будет использоваться в таблице для определения, о том какую информацию и в каком порядке показывать пользователю.

Следующая функция: вывод информации о всём доступном транспорте по нажатию кнопки. Для этого создаётся объект класса Button и к нему привязывается метод объекта класса table, который запрашивает у базы данных информацию о всём транспорте, применяет выбранные фильтры и сортировки и отображает информацию в таблице.

После вызова данных и их фильтрации окно программы выглядит так (см. рисунок 4):

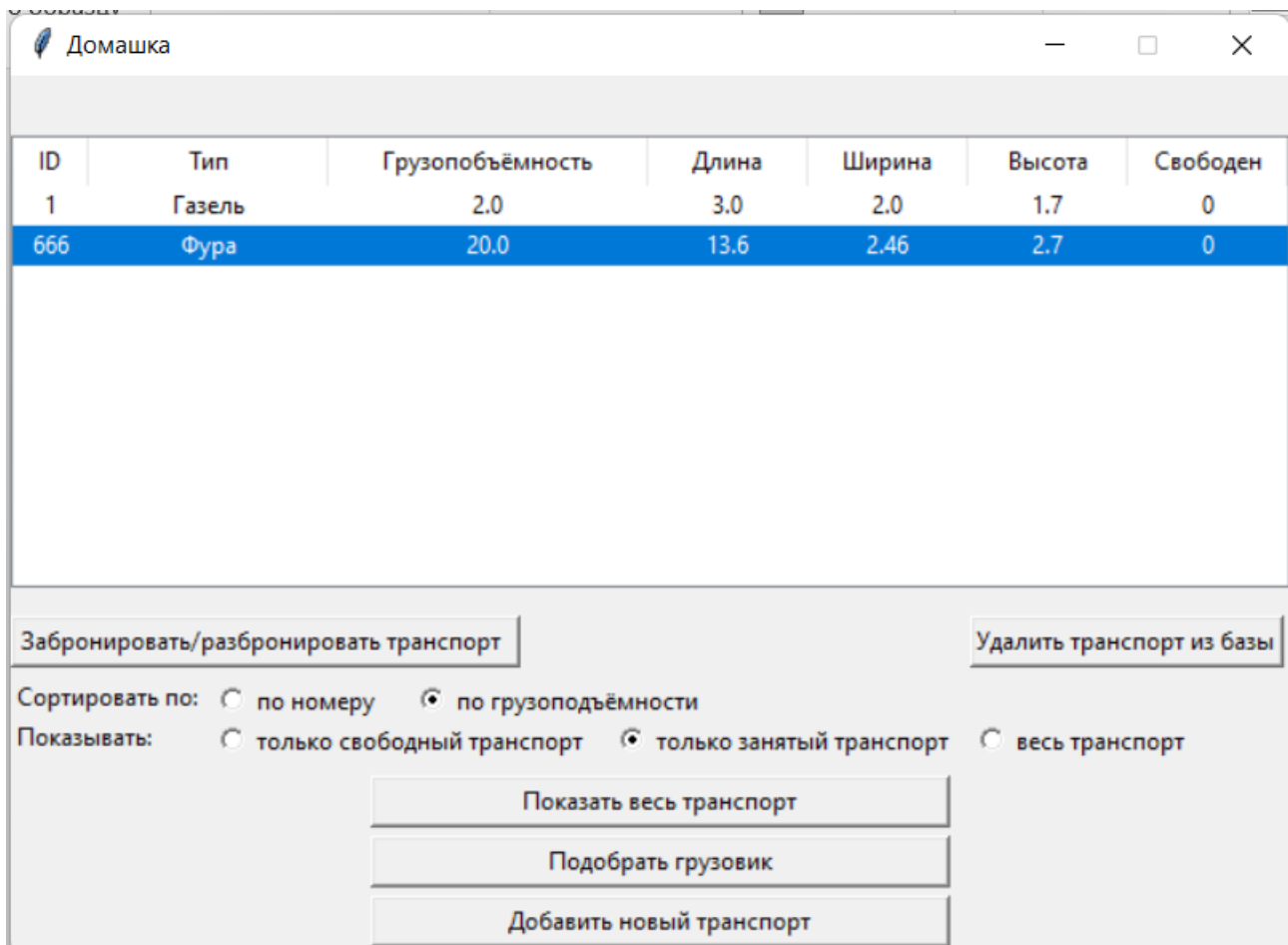


Рисунок 4. Интерфейс программы после вызова информации о всех занятых машинах.

После нажатия кнопки на экране появляется запрошенная информация, теперь пользователь может просматривать доступный транспорт и бронировать нужный ему.

Для того чтобы получить от пользователя информацию о грузе создаётся всплывающее окно (см. рисунок 5):

```
self.entry_window = Toplevel()
self.entry_window.title('Окно ввода')
self.entry_window.geometry('370x140')
self.entry_window.resizable(0, 0)
self.l_weight = Label(self.entry_window, text='Введите массу перевозимого груза')
self.l_length = Label(self.entry_window, text='Введите длину перевозимого груза')
self.l_width = Label(self.entry_window, text='Введите ширину перевозимого груза')
self.l_height = Label(self.entry_window, text='Введите высоту перевозимого груза')
k = 0

if self.mode == 1:
```

Рисунок 5. Создание всплывающего окна.

Чтобы подобрать машину по параметрам нужно забросить у пользователя информацию о перевозимом грузе. Для этого создаётся вспомогательное всплывающее окно для ввода информации, оно создаётся как отдельный класс. За основу берётся виджет `Toplevel`. При этом, чтобы пользователь не мог вызвать

слишком много таких окон, на время его работы, кнопки в основном окне блокируются. Далее на всплывающее окно добавляются пояснения о том, какая информация запрашивается (виджет Label), а также поля для ввода этой информации (виджет Entry). После того как пользователь подтверждает ввод (для этого создаётся кнопка), его ввод проверяется и в случае ошибок вызывается ошибка и пользователю предлагается попробовать ввести информацию снова. Если введена неполная информация выводится предупреждение, но программа продолжает работу с той информацией, что была введена.

Так будет выглядеть диалог с пользователем (см. рисунок 6):

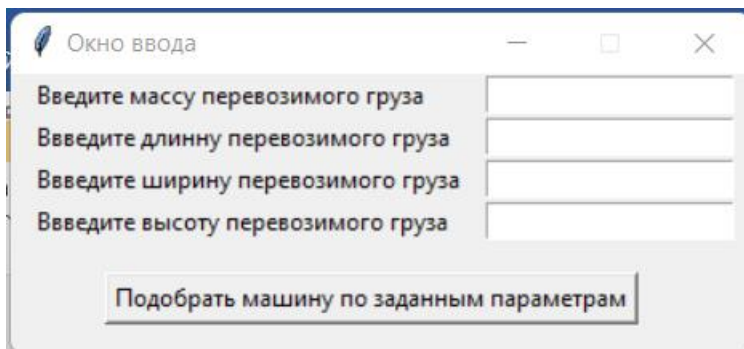


Рисунок 6. Окно ввода дополнительной информации о грузе.

После успешной проверки информация о транспорте фильтруется и пользователю предлагаются только подходящие модели, что заметно облегчает поиск.

Добавление нового транспорта работает схожим образом, также вызывается всплывающее окно, но теперь появляются 2 новых поля. Также проверка ввода более строгая: необходима полная информация, иначе пользователю будет предложено дополнить свой ввод. После успешного ввода обновляется база данных, и новая машина вскоре появляется на экране пользователя.

Так будет выглядеть диалог с пользователем (см. рисунок 7):

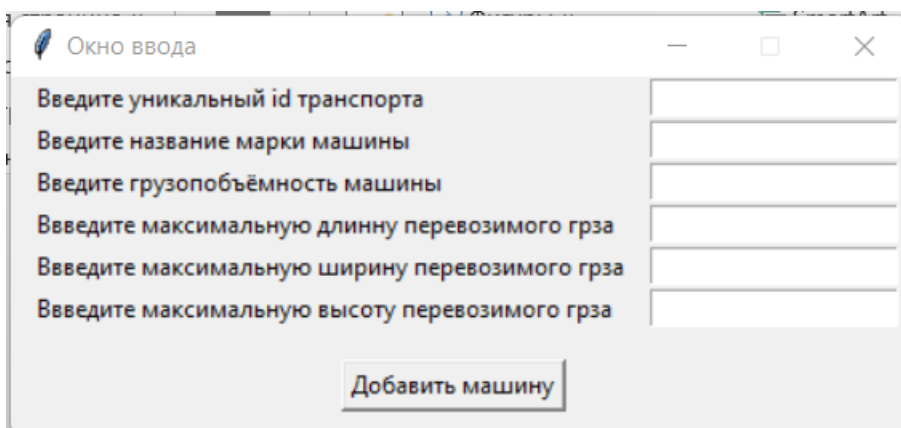


Рисунок 7. Окно добавления нового транспорта.

Вывод:

Средствами библиотеки tkinter была создана программа с графическим интерфейсом, помогающая пользователю вести учёт грузового транспорта. GUI программы представляет из себя набор из виджетов классов Button, Treeview,

RadioButton, Label и Entry. С помощью библиотеки sqlite3 была добавлена работа с базами данных, что позволяет пользователю сохранять изменения, внесённые в данные и возвращаться к ним после завершения работы с программой. Также была предусмотрена обработка исключений, поэтому программа будет продолжать штатно работать, даже в случае, если пользователи будут допускать ошибки во взаимодействии с программой.