Оглавление

[Введение **2**](#_Toc27597673)

[**1.** **Теоретическая часть** **3**](#_Toc27597674)

[1.1 Исследование предметной области 3](#_Toc27597675)

[1.2 Выбор структур данных для реализации операций хранения, сортировки и поиска 4](#_Toc27597676)

[1.4 Вывод к первой главе 12](#_Toc27597677)

[**2.** **Проектная часть** **12**](#_Toc27597678)

[2.1. Разработка пользовательского интерфейса 12](#_Toc27597679)

[2.2. Проектное решение 12](#_Toc27597680)

[**2.2.1.** **Разработка спецификации** **12**](#_Toc27597681)

[2.3. Реализация программного продукта 13](#_Toc27597682)

[**2.3.1.** **Реализация возможности хранения** **13**](#_Toc27597683)

[**2.3.2.** **Реализация программного изделия** **13**](#_Toc27597684)

[2.4. Описание структуры интерфейса пользователя 14](#_Toc27597685)

[2.5. Вывод ко второй главе 19](#_Toc27597686)

[**3.** **Экспериментальная часть** **19**](#_Toc27597687)

[3.1. Цель испытаний 19](#_Toc27597688)

[3.2. Тестирование 19](#_Toc27597689)

[Заключение 29](#_Toc27597807)

[**Список используемых источников** **30**](#_Toc27597808)

[**Приложение** **31**](#_Toc27597809)

# Введение

В данной курсовой работе представлена реализация програмного приложения «Гаражная стоянка 2.0». В нее входят система парковки автомобилей (въезд на стоянку, выезд со стоянки), система модерироваиня множества стоянок (создание и удаление стоянок с разными характеристиками, въезд и выезд с определённой стоянкеи), сортировка автомобилей на выбраной парковке, учёт всех автомобилей вьезжающих и выезжающих со стоякнки, учёт автомобилей (во время парковки и выезда каждый автомобиль заноситься в базу с учётом время выезда и въезда), попадающих в VIP список (добавление и удаление автомобиля из VIP списока).

Программная часть реализована при помощи языка Python, а также графического фреймворка Qt в среде разработки PyCharm.

Ведение парковочной деятельности – сложный процесс, требующий внимания и повышенной ответственности. Программное приложение «Гаражная стоянка 2.0» даёт возможность удобного учёта припаркованных автомобилей. Оно исключает проблем с расхождение свободных мест по подсчёту и реальных свободных мест.

Данное приложение актуально сейчас, как никогда – в наше время идет активное развитие информационных технологий в компаниях, занимающихся ведением парковочной деятельности, поэтому появление подобного ПО может заменить ручной учет составляющих этого бизнеса что ползволит ускорить деятельность занимавшую значительное время при ручном труде.

1. **Теоретическая часть**
   1. **Исследование предметной области**

При исследовании предметной области встал вопрос о том, как можно максимально упростить для пользователей использование разрабатываемого приложения.

Отталкиваясь от темы курсовой работы, было принято решение реализовать информационную систему как «Desktop application».

При анализе предметной области были выявлены основные подразделения охвата компании и их критерии:

1. **Парковки**

Парковка подразумевает под собой место стояники автомобилей, с возможностью их добавления (с опредеёнными характеристиками), удаления. Характеристика парковки подразумевает:

* + Имя;
  + Максимальное кол-во обычных мест;
  + Максимальное кол-во VIP мест.

Автомобиль включает в себя несколько основных характеристик. При добавлении автомобиля для него из индивидуального регистрационного номера генерируеться ID, который исспользуестья в операциях сортировки автомобилей на парковке.

Характеристики автомобия:

* + Цвет;
  + Регистрационный номер;
  + ID.

1. **VIP список**

Область, позволяющая добавлять и удалять индивидуальные регистрационные номера автомобилей. Данная область представляет из себя базу данных в которой храняться характеристики автомобиля, относящегося к VIP категории.

Характеристики автомобля:

* + Индивидуальный регистрационный номер
  + ID

1. **Клиентская база данных**

Клиенская база данных нужна, для предаставления индивидуальных условий для пользователей стоянки исходя из посещения данной стоянки.

Характеристики, хранящиеся в базе данных:

* + Индивидуальный регистрационный номер автомобиля;
  + ID;
  + Время въезда;
  + Время выезда.

При исследовании предметной области также был определен основная задача приложения. Она состоит в упрощении учетной деятельности стояночных парковщиков.

* 1. **Выбор структур данных для реализации операций хранения, сортировки и поиска**

Данная работа предполагает 3 вида структур данных: стек, двоичное дерево поиска и список. Каждая структуда данных предназначена для различного исспользования и используеться в различных хранилищах данных, таких как автомобиль и стояника.

Разберем поочередно каждое из хранилищ, и, исходя из требований, выберем нужную структуру.

Первая структура предназначена для хранения информации о автомобиле, она исспользует стандартные структуры такие как строка и число, для хранения данных о автомобиле.

Вторая структура предназначена для хранения информации о нескольких автомобилях, информация о их парковке.

Автомобиль в нашем приложении являеться основным обьектом, вокруг кооторого строиться приложение. Нам необходимо предусмотреть возможность заезда и выезда автомобился с парковки. А также сортировку автомобилей на парковке. Мы можем сделать вывод, что для операция должны исспользоваться разные структуры данных. Мы будем исспольовать такие структуры: двоичное дерево поиска, стек и лист. Стек будет исспользоваеться как основная структура данных, в которой будет храниться информациия о нахождении каждого автомобиля на парковке. Двоиченое дерево поиска будет исспользовано для сортировки автомобилий находящихся на парковке в структуре стек.

**Двоичное дерево поиска** — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Так же для двоичного дерева поиска должны выполняться следующие условия:

* Оба поддерева (левое и правое) — являються двоичными деревтями поиска;
* У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, неженли значение ключа данных самого узла X;
* У всез узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

Ниже на рисунке представлено графическое изображение двоичного дерева поиска.

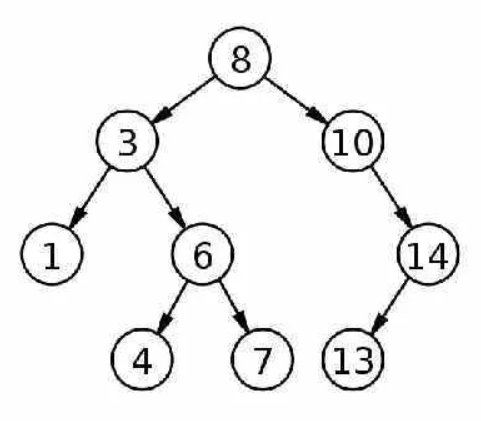


Рисунок 1 — Графическое изображение двоичного дерева поиска

**Стек** — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO. Чаще всега принцип работа стака стравнивают со стопкой йтарелок: чтобы взхять вторую сверху тарелку, нужно снять вернюю.

Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями, также в отличии от массива в список легко добавить элемент в середину, без тяжелых или цикличных перестроений. Ниже на рисунке представлено графическое изображение линейного односвязного списка.

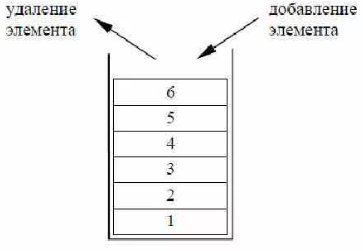


Рисунок 2 — Графическое изображение стека

После интеграции в систему данной структуры к ней были применены изменения, создан метод, который возврящает количество элементов, находящихся в стеке, для ускорения работы программы.

**1.3. Описание алгоритмов, применяемых к выбранным структурам данных**

В данной работе представлено 3 различных структуры данных с уникальными функциями добавления, удаления, поиска и т.п. Далее мы разберем конкретный структуры с их функционалом

* + 1. **Стек**

**Код структуры данных**

class Stack :  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()  
  
 def is\_empty(self):  
 return (self.items == [])  
  
 def size(self):  
 return len(self.items)

**Добавление**

Функция добавления «push», получает на вход объект класса. Далее она кладёт элимент на стек, примерно так же как это было описано в примере с тарелками. Тоесть обьект полученый функцией добавление становиться верхним эллиментом.

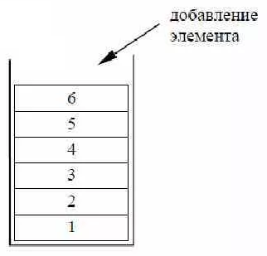


Рисунок 3 — Графическое изображение добавление элемента в стек

**Удаление**

Функция удаления «pop», возвращает последний добавленый в стек элемент, удаляя (вытаскивая) его из стека.

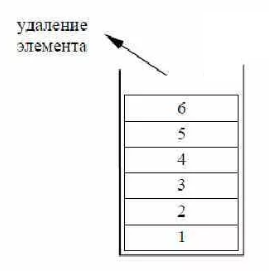


Рисунок 4 — Графическое изображение удаления элемента из стека

**Вывод хранящихся эллементов в той же последовательности**

Функция вывода элементов, хранящихся в исходном стеке, изменяя последовательность в изначальном стеке изначально проходиться по всему стеку, отправляя все элементы в буферный стек, что отзеркаливает последовательность эллементов, в этот же момент аглоритм записывает автомобильные номера в спислок. После того как все эллементы из начального стека были перемещенны в буферный стек и записаны в список, элементы буфферного списка отправляються обратно в изначальный стек. Это сохраняет последовательность изначального стека, не теряя элементов и не меняя их раположение. После возвращения исходного стека в изначальный вид функция возвращает список с автомобильными номерами в нужной последовательности.

Описываемый алгоритм:

def PrintStackDef(self):  
 BufStack = Stack()  
 list = []  
 for i in range(self.parkDef.size()):  
 car = self.parkDef.pop()  
 BufStack.push(car)  
 list.append(car.RegNum)  
 for i in range(BufStack.size()):  
 car = BufStack.pop()  
 self.parkDef.push(car)  
 return list

**Выезд автомобиля со стоянки**

Функция выезда автомобился со стоянки «DelCar», если автомобиль был найдет на парковке, то функция возвращает количество выездов автомобиля для обеспечения выезда других автомобилей со стоянки, иначе возвращаеться значение None, что соотвестствует тому, что автомобиля на парковке не было. Алгоритм проходиться по всему стеку, сравнивая каждый автомобильный номер, с тем номером, который мы ввели что бы данный автомобиль выехал с парковки, если номера не соответсвуют, то автомобиль отправляетьтся в буферный стек и количество выездов автомобиля увеличиваеться. Если автомобиль соответствует нашему, то обновяляются таблицы в базах данных и машины не записываеться в буферный стек. После того как исходный стек остался пустым все элементы буферного стека возвращаются обратно в исходный стек. В итоге стек парковки оставляет исходную последовательность без сохранения выехавшего автомобиля.

Описываемы алгоритм:

def DelCar(self, car):  
 BufStack = Stack()  
 CarHere = False  
 intMoved = None  
 for i in range(self.parkDef.size()):  
 newCar = self.parkDef.pop()  
 if(newCar.RegNum == car.RegNum): # Удаление машины  
 CarHere = True  
 intMoved = newCar.Moved  
 SetUnparking(None, car)  
 AddFreePlaceDef(-1, self.Name)  
 else:  
 newCar.Moved += 1  
 BufStack.push(newCar)  
 for i in range(BufStack.size()):  
 self.parkDef.push(BufStack.pop())  
  
 if(CarHere):  
 return intMoved  
  
 for i in range(self.parkVIP.size()):  
 newCar = self.parkVIP.pop()  
 if(newCar.RegNum == car.RegNum): # Удаление машины  
 CarHere = True  
 intMoved = newCar.Moved  
 SetUnparking(None, car)  
 AddFreePlaceVIP(-1, self.Name)  
 else:  
 BufStack.push(newCar)  
 for i in range(BufStack.size()):  
 self.parkVIP.push(BufStack.pop())  
  
 if (CarHere):  
 return intMoved

* + 1. **Двоичное дерево поиска**

**Вставка нового элемента в двоичное дерево поиска**

Рекурсивная функция, добавляющая новый элемент в дерево. Имеет три исхода развития событий:

* + - * 1. Дерево отсутствует или веток больше не найдено. Это выходной из рекурсии сценарий, при котором добавляется элемент в текущую позицию.
        2. Добавляемы элемент меньше текущего. В этом случае выбирается левая ветка.
        3. Добавляемы элемент больше или равен текущему. В этом случае выбирается правая ветка.

Алгоритм вставки элемента в двоичное дерево поиска:

def insert(tree, key):  
 if not tree:  
 tree = Node\_tree(key)  
 elif key < tree.key:  
 tree = Node\_tree(tree.key, insert(tree.left, key), tree.right)  
 elif key > tree.key:  
 tree = Node\_tree(tree.key, tree.left, insert(tree.right, key))  
 else:  
 tree = Node\_tree(tree.key, tree.left, insert(tree.right, key))  
 return tree

**Вывод двоичного дерева поиска**

Алгоритм «postorder» проходиться по всему дереву исспользуя рекурсию. Он проходиться изначально по левым веткам, после по правым, записываея значения в список «temp».

Описываемый алгоритм «postorder»:

def postorder(tree, temp):  
 if tree:  
 if(tree.left!=None):  
 postorder(tree.left, temp)  
 if(tree.right!=None):  
 postorder(tree.right, temp)  
 temp.append(tree.key)

**Сотрировка стоянки деревом**

Сортировки стоянки деревом, из списка со всеми припарковаными автомобилями ID записываються в двоичное дерево поиска, после того как все ID будут записаны в дерево мы получаем отсортированный список всех ID исспльзуя алгоритм «postorder». Далее мы из листа автомобилей находим автомобиль с номером как в отсортированном списке и записываем автомобиль обратно в парковочный стек, так и получаеться отсортированный стек.

Алгоритм сортировки стоянки деревом:

def SortTree(self):   
 temp = []   
 ListDef = self.ReturnCarListDef()  
 TreeDef = None  
 for car in ListDef:   
 TreeDef = insert(TreeDef, car.ID)  
 postorder(TreeDef, temp)   
 self.parkDef = Stack()   
  
 for carID in temp:   
 for car in range(len(ListDef)):   
 if(carID == ListDef[car].ID):   
 self.parkDef.push(ListDef[car])  
 ListDef.pop(car)  
 break

* 1. **Вывод к первой главе**

В результате анализа предметной области, структур данных, алгоритмов сортировки и поиска была построена структура всего проекта.

1. **Проектная часть**
   1. **Разработка пользовательского интерфейса**

Проектная часть курсового проекта описывает задачи связанные с проектированием. К таким задачам относятся:

* формирование общих требований;
* определение функциональной части проекта;
* проектирование графического интерфейса пользователя.
  1. **Проектное решение**

Система «Гаражная стоянка Гаражная стоянка 2.0» разработана как приложение для компьютеров, с удобным пользовательским интерфейсом. Данные хранятся в файлах с расширением .sqllite.

Основным языком реализации является Python (по заданию на курсовой проект). Также при разработке использовались такие инструментальные средства:

* PyQt5;
* Qt Designer.
  + 1. **Разработка спецификации**

В качестве основной спецификацией будет являться являться функционал, который данное приложени должно выполнять. Данный функционал является основополагающим: он задает основыне свойства системы и в нем заключены ее возможности. К таким функциональным требованиям относятся:

* Вывод всех таблиц базы данных;
* Создание парковок с изменением кол-ва мест;
* Удаление парковок;
* Добавление автомобиля в VIP лист;
* Удаление автомобиля из VIP листа;
* Вывод информации о припаркованых автомобилях без изменения послодовательности;
* Парковка автомобилей с выбором парковки;
* Выезд с парковки;
* Сортировка автомобилей на парковках;
* Изменения информации в базе данных.

Методы классов структур хранения, полная их декомпозиция находится в разделе «Описание алгоритмов, применяемых к выбранным структурам данных»

* 1. **Реализация программного продукта**
     1. **Реализация возможности хранения**

Все данные, хранящиеся в программе, так же сохраняються в базу данных, реализованную на основе SQLlite.

Предусмотрена одна база данных хранящая в себе три разных таблицы: которые реализуют сохраниние VIP автомобилей, сохранение стоянок и информации о них и информацию о въезде и выезде автомобиля на с парковки.

При открытии приложения из базы данных загружаеться вся информация о парковках.

При каких либо дейсвия связанных с изменением базы данных, то данные в базе данных меняються.

* + 1. **Реализация программного изделия**

Во время разработки следует разбить програмное обеспечение на несколько областей по смыслу, для удобства пользования.

Пользовательский ввод должне проверяться на основе допустимых значений и выводить ошибку если что-то пошло не так.

Для того чтобы точнее проинформировать пользователя о неккоректном вводе, следует разбить одну большую проверку, на несколько мнеьших, что позволит увеличить разннообразие отображаемых сообщений с предупреждением об ошибке ввода или другой ошибке.

* 1. **Описание структуры интерфейса пользователя**

Пользовательский графический интерфейс был спроектирован с условием удобства и интуетивности при исспользовании. По причине широково функуионала приложени и большого колическа элементов, мной было принято решение разделить функционал приложения на основные группы, иссползолуя элемент «Group box»

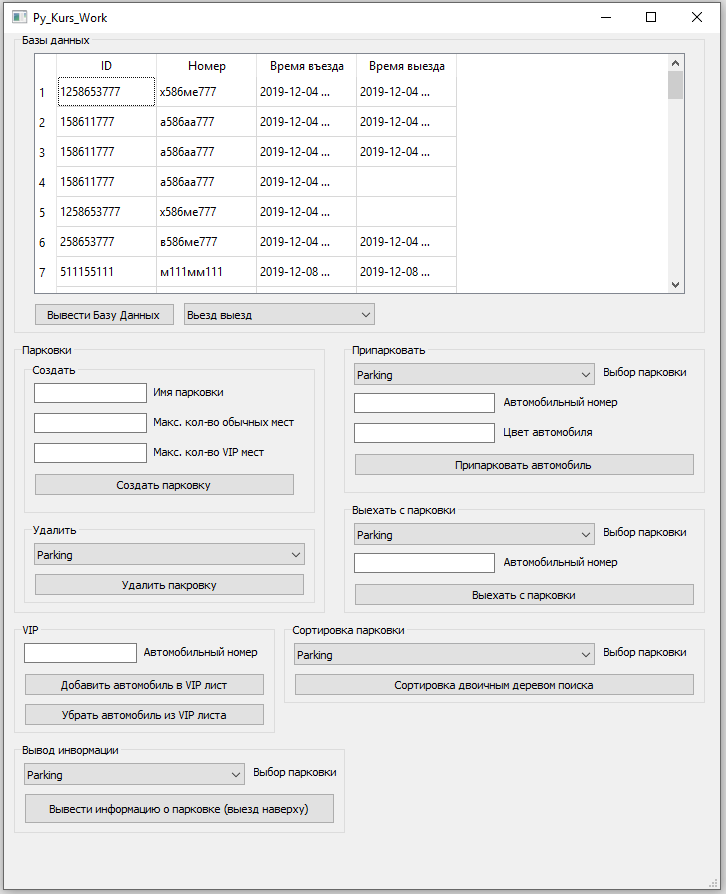


Рисунок 5 — Интерфейс программы (БД Въезда и выезда)

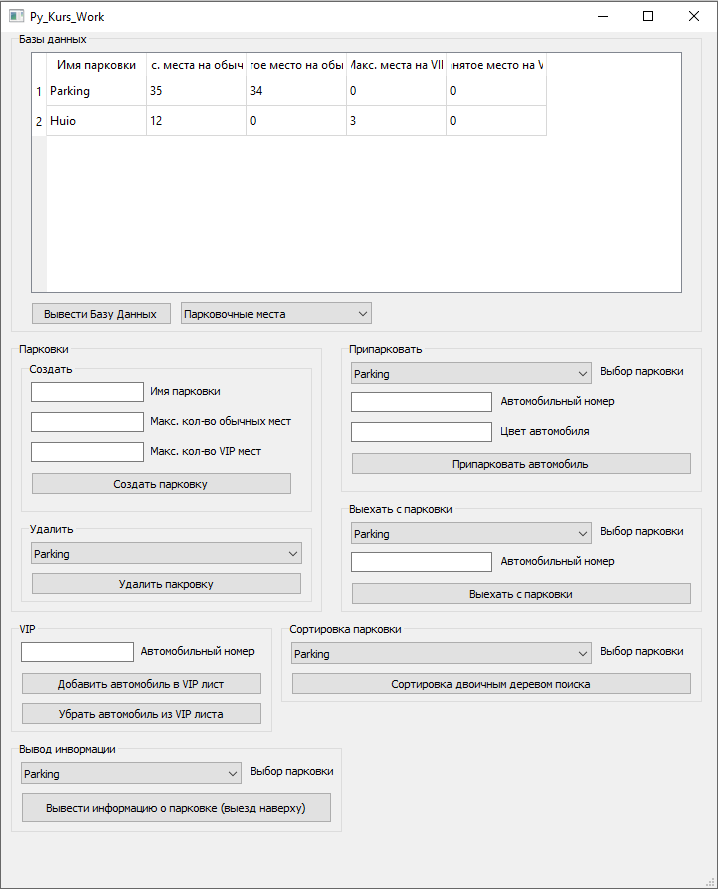


Рисунок 6 — Интерфейс программы (БД парковок с местами)

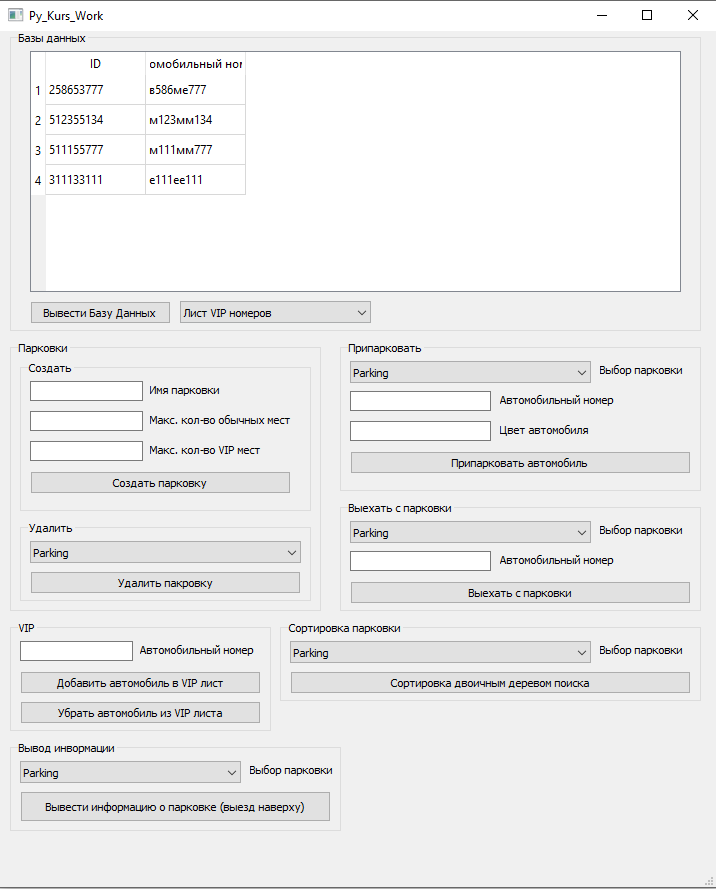


Рисунок 8 — Интерфейс программы (БД VIP автомобили)

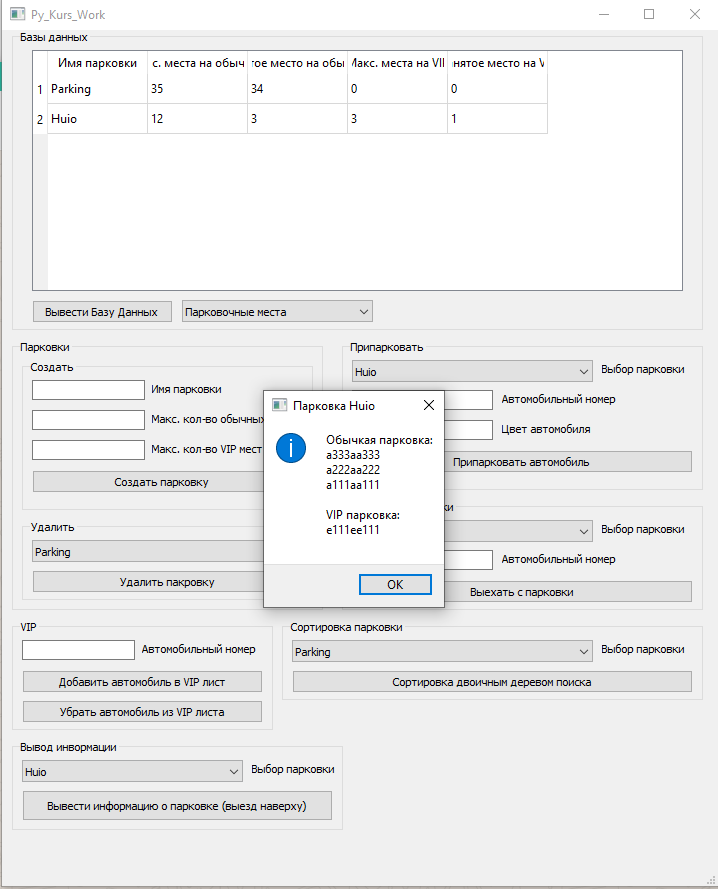


Рисунок 9 — Интерфейс программы (Информация о припаркованных)

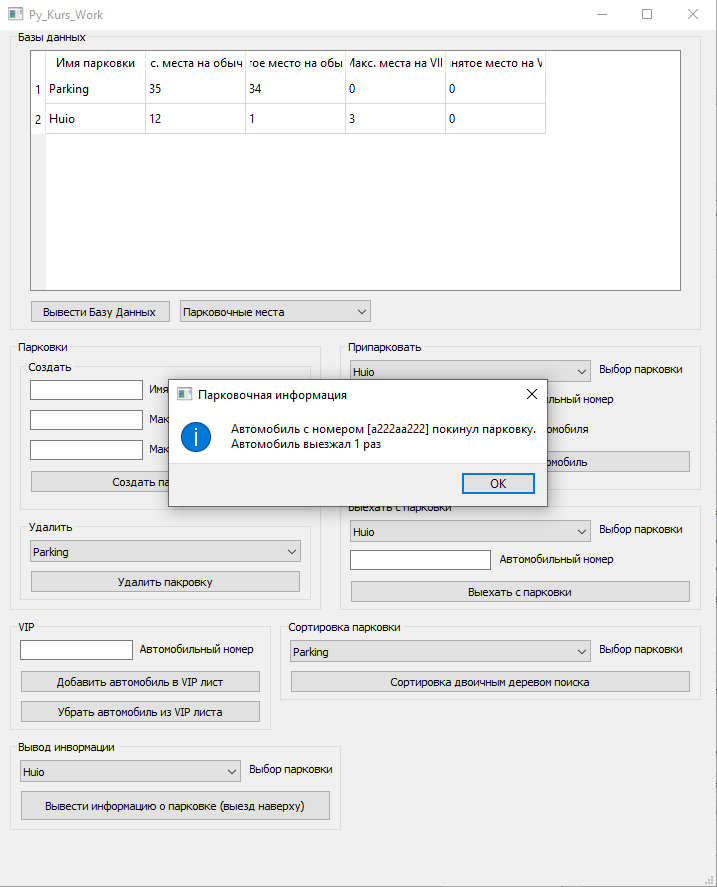


Рисунок 9 — Интерфейс программы (Выезд с парковки)

* 1. **Вывод ко второй главе**

В результате выполнения работа по проектной части курсовой, удалось определить функционал приложения и спроектировать пользовательский интеряейс проекта.

1. **Экспериментальная часть**
   1. **Цель испытаний**

Целью проведения испытаний является:

* Проверка работоспособности функция продукиа;
* Проверка соответствия заявленным характеристикам и требованиям из технического задания;
* Проверка готовности программы к проведению приёмочных испытаний;
* Определение уязвимостей проекта и дальнейшее их устранение.
  1. **Тестирование**

|  |  |
| --- | --- |
| Тестирование добавления парковки | |
| Ввод |  |
| Вывод | Сообщени о неправильном вводе. |
| Ожидаемый вывод | Сообщени о неправильном вводе. |

Таблица 1 — Тестирование добавления парковки

|  |  |
| --- | --- |
| Тестирование добавления парковки | |
| Ввод |  |
| Вывод | Появление парковки в базе данных. |
| Ожидаемый вывод | Появление парковки в базе данных. |

Таблица 2 — Тестирование добавления парковки

|  |  |
| --- | --- |
| Тестирование удаления парковки | |
| Ввод | Пустая парковка «ParkName» |
| Вывод | Удалени парковки из базы данных. |
| Ожидаемый вывод | Удалени парковки из базы данных. |

Таблица 3 — Тестирование удаления парковки

|  |  |
| --- | --- |
| Тестирование удаления парковки | |
| Ввод | Не пустая парковка «ParkName» |
| Вывод | Вывод ошибки: парковка не пустая. |
| Ожидаемый вывод | Вывод ошибки: парковка не пустая. |

Таблица 4 — Тестирование удаления парковки

|  |  |  |
| --- | --- | --- |
| Тестирование добавления в VIP | | |
| Ввод | Авто номер котого нет в VIP. | Авто номер который есть в VIP. |
| Вывод | Появление в базе данных. | Вывод ошибки. |
| Ожидаемый вывод | Появление в базе данных. | Вывод ошибки. |

Таблица 5 — Тестирование добавления в VIP

|  |  |  |
| --- | --- | --- |
| Тестирование удаления из VIP | | |
| Ввод | Авто номер котого нет в VIP. | Авто номер который есть в VIP. |
| Вывод | Вывод ошибки. | Удаления из базы данных. |
| Ожидаемый вывод | Вывод ошибки. | Удаления из базы данных. |

Таблица 6 — Тестирование удаления из VIP

|  |  |  |
| --- | --- | --- |
| Тестировани вывод парковки | | |
| Ввод | Пустая парковка Parking | Не пустая парковка. |
| Вывод |  |  |
| Ожидаемый вывод |  |  |

Таблица 7 — Тестироване вывод парковки

|  |  |  |
| --- | --- | --- |
| Тестирование въезд на парковку | | |
| Ввод |  |  |
| Вывод |  |  |
| Ожидаемый вывод |  |  |

Таблица 8 — Тестирование въезд на парковку

|  |  |
| --- | --- |
| Тестирование въезд на заполненую парковку | |
| Ввод |  |
| Вывод |  |
| Ожидаемый вывод |  |

Таблица 9 — Тестирование въезд на заполненую парковку

|  |  |  |
| --- | --- | --- |
| Тестирование выезд с парковки | | |
| Ввод |  |  |
| Вывод |  |  |
| Ожидаемый вывод |  |  |

Таблица 10 — Тестирование выезд с парковки

|  |  |
| --- | --- |
| Тестирование сортировка парковки деревом | |
| Ввод | Не пустая парковка «Huio» |
| Вывод |  |
| Ожидаемый вывод |  |

Таблица 11 — Тестирование сортировка парковки деревом

Тестовые запуски показали, что приложение:

* Работоспособно;
* Устойчиво к ошибкам пользователя;
* Выполнено в соответствие с требованиями.

**Заключение**

В результате выполнения курсовой работы, была спроектирована и разработана программная система «Гаражная стоянка 2.0» реализующая работу.

Реализованная система имеет широкий ряд функциональных возможностей.

В процессе работы над курсовой работой были закреплены навыки:

* Анализа предметной области;
* Проектирования, разработки баз данных;
* Разработки приложений на языке Python;
* Разработки приложения с проектированием пользовательского интерфейса с исспользованием Qt;
* Составления SQL запросов;
* Тестирования программных компонентов;
* Работы с базами данных, таких как SQLlite;
* Подготовки технической документации.

**Список используемых источников**

Нормативные документы:

1. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Стандартинформ, 2017. – 32 с.;
2. О введении в действие Инструкции по организации и проведению курсового проектирования. – М.: РТУ МИРЭА, Приказ №1325 от 05.10.2018. – 17 с.;

Электронные ресурсы:

1. Руководство по PyQt5 [Электронный ресурс]. – URL: <https://python-scripts.com/pyqt5#qcombobox-in-qtablewidget>
2. Python: Работа с базой данных, часть 1 [Электронный ресурс]. – URL: <https://habr.com/ru/post/321510/> (11.02.2017)
3. Базы данных в Python [Электронный ресурс]. – URL: <https://python-scripts.com/database>

**Приложение**

**ЛИСТИНГ**

**Файл со структурой данных двоичное дерево поиска**

class Node\_tree:  
 def \_\_init\_\_(self, key, left=None, right=None):  
 self.key = key  
 self.left = left  
 self.right = right  
  
 def \_\_str\_\_(self):  
 return "(%s %s %s)" % (self.left or "", self.key, self.right or "")  
  
  
def insert(tree, key):  
 if not tree:  
 tree = Node\_tree(key)  
 elif key < tree.key:  
 tree = Node\_tree(tree.key, insert(tree.left, key), tree.right)  
 elif key > tree.key:  
 tree = Node\_tree(tree.key, tree.left, insert(tree.right, key))  
 else:  
 tree = Node\_tree(tree.key, tree.left, insert(tree.right, key))  
 return tree  
  
def postorder(tree, temp):  
 if tree:  
 if(tree.left!=None):  
 postorder(tree.left, temp)  
 if(tree.right!=None):  
 postorder(tree.right, temp)  
 temp.append(tree.key)

**Файл car.py**

import datetime  
import re  
class Car(object):  
  
 Liters = ['а', 'в', 'е', 'к', 'м', 'н', 'о', 'р', 'с', 'т', 'у', 'х'] # Все буквы из автомобильных номеров  
 Color = ""  
 RegNum = "" # Номер выглядит так wNNNwwNNN  
 ID = None # Переменная равна None, кодга есть проблемы с созданием экземпляра класса Car  
 Moved = 0 # Количество выездов, что бы освободить выезд с парковки  
  
 def \_\_init\_\_(self, Color, RegNum, Moved = 0):  
 if(isinstance(Color, str) and isinstance(RegNum, str) and re.match("\w{1}\d{3}\w{2}\d{2,3}", RegNum) and len(RegNum) < 10): # Проверяет соответствует ли номер данному патрену (номер РФ). А так же проверяте что имено передавалось  
 try:  
 self.Color = Color  
 self.RegNum = RegNum  
 self.SetId()  
 self.Moved = Moved  
 except Exception as e:  
 ID = None  
 else:  
 ID = None  
  
 def SetId(self): # Задаёт ID, получает его изменением РегНомера. Для сотрировки.  
 RegNumber = self.RegNum  
 for liter in range(0, len(self.Liters)):  
 RegNumber = RegNumber.replace(self.Liters[liter], str(liter + 1))  
 self.ID = int(RegNumber)

**Файл Stack.py**

class Stack :  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()  
  
 def is\_empty(self):  
 return (self.items == [])  
  
 def size(self):  
 return len(self.items)

**Файл DataBase.py**

import sqlite3  
from Car import Car  
import datetime  
  
#region Information  
# FROM выбирает таблицу  
# SELECT вобор столба нужного мне  
# WHERE условие для вывода SELECT  
  
DataBaseName = "DataBase.sqllite"  
TableName = "CarParkingInfo"  
NameListVIP = "VIP\_List"  
NameParkList = "Places"  
#endregion  
  
#region CarParkingInfo  
def AddToDataBase(Car, ParkTime): # Добавляет регНомер и время в БД  
 try:  
 DBconnect = sqlite3.connect(DataBaseName) # Подключение к базе (возмжоно нужно ватащить наружу)  
 cursor = DBconnect.cursor() # Место нахождение в таблице  
 if(ParkTime == None): # Если время парковки НОНЕ, то автоматом задаётся дата на данный момент  
 ParkTime = str(datetime.datetime.now())  
 cursor.execute("INSERT INTO {} (ID, RegNumber, ParkingTime, OutTime) VALUES ('{}', '{}','{}', '{}')".format(TableName, Car.ID, Car.RegNum, ParkTime, ' '))  
 DBconnect.commit() # Синхронизирует изменения с баззой  
 DBconnect.close() # Закрывает открытую БД  
 except Exception as e:  
 print("Ошибка при занесении данных в базу: " + e)  
 DBconnect.close()  
 return None  
  
def GetTime(Car): # Пока хз зачем это  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT ParkingTime FROM {} WHERE RegNumber='{}'".format(TableName, Car.RegNum))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return res  
 except Exception as e:  
 print("Ошибка при получении времени из базы: " + e)  
 DBconnect.close()  
 return None  
  
def DelFromDataBase(Car): # Удаляет каждое упоминание о номере  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("DELETE FROM {} WHERE RegNumber='{}'".format(TableName, Car.RegNum))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при удалении из базы: " + e)  
 DBconnect.close()  
 return None  
  
def SetUnparking(UnParkingTime, Car): # Задаёт время выеза, примерно так же как и время вьезда  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 if(UnParkingTime == None):  
 UnParkingTime = str(datetime.datetime.now())  
 # cursor.execute("SELECT ParkingTime FROM {} WHERE RegNumber='{}'".format(TableName, RegNumber))  
 # res = cursor.fetchall()  
 # print(res[0][0]) # Получает строку времени парковки  
 cursor.execute("UPDATE {} SET OutTime='{}' WHERE RegNumber='{}' AND OutTime=' '".format(TableName, UnParkingTime, Car.RegNum))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при задавании времени выезда в базу: " + e)  
 DBconnect.close()  
 return None  
  
def CarOnParing(Car): # Если есть пустой АутТайм, то автомобиль ещё на паркинге  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT OutTime FROM {} WHERE RegNumber='{}'".format(TableName, Car.RegNum)) # Получаем все АутТаймы, где данный номер  
 res = cursor.fetchall() # Если в АутТайме есть пустой эллемент, это значит что автомобиль находиться на парковке  
  
 result = False  
 if(len(res) == 0): # Если нет эллементов АутТайм, то тачка точно не на парковке  
 result = False  
 else: # Проверяет все элементы АутТайм и если какой то из них равен пробелу, тачка на парковке  
 for i in res:  
 if(i[0] == " "):  
 result = True  
  
 DBconnect.close()  
 return result  
 except Exception as e:  
 print("Ошибка при поиске наличии машины на парковке: " + e)  
 DBconnect.close()  
 return e  
#endregion  
  
#region VIP List  
def VIP(Car): # Если машина в вип листе  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT RegNum FROM {}".format(NameListVIP))  
 res = cursor.fetchall()  
  
 for i in range(0, len(res)):  
 if(res[i][0] == Car.RegNum):  
 DBconnect.close()  
 return True  
  
 DBconnect.close()  
 return False  
 except Exception as e:  
 print("Ошибка при проверки ВИП автомобиля: " + e)  
 DBconnect.close()  
 return None  
  
def AddToVIP(Car): # Добавить в VIP лист  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 if(VIP(Car) == False):  
 cursor.execute("INSERT INTO {} (RegNum, ID) VALUES ('{}', '{}')".format(NameListVIP, Car.RegNum, Car.ID))  
  
 DBconnect.commit()  
 DBconnect.close()  
 return False  
 except Exception as e:  
 print("Ошибка при добавлении в VIP список: " + e)  
 DBconnect.close()  
 return None  
  
def DellFromVIP(Car):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("DELETE FROM {} WHERE RegNum='{}'".format(NameListVIP, Car.RegNum))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при удалении из базы: " + e)  
 DBconnect.close()  
 return None  
#endregion  
  
#region Places  
#region Get  
def GetMaxPlaceDef(ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("MaxPlaceDef", NameParkList, ParkName))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return int(res[0][0])  
 except Exception as e:  
 print("Ошибка при получении максимального места на стандартной парковке: " + e)  
 DBconnect.close()  
 return None  
  
  
def GetMaxPlaceVIP(ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("MaxPlaceVIP", NameParkList, ParkName))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return int(res[0][0])  
 except Exception as e:  
 print("Ошибка при получении максимального места на vip парковке: " + e)  
 DBconnect.close()  
 return None  
  
def GetFreePlaceDef(ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("FreePlaceDef", NameParkList, ParkName))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return int(res[0][0])  
 except Exception as e:  
 print("Ошибка при получении занятого места на обычной парковке: " + e)  
 DBconnect.close()  
 return None  
  
def GetFreePlaceVIP(ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("FreePlaceVIP", NameParkList, ParkName))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return int(res[0][0])  
 except Exception as e:  
 print("Ошибка при получении занятого места на vip парковке: " + e)  
 DBconnect.close()  
 return None  
  
def GetAllParkings():  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT \* FROM {}".format(NameParkList))  
 res = cursor.fetchall()  
 DBconnect.close()  
 return res  
 except Exception as e:  
 print("Ошибка при получении занятого места на vip парковке: " + e)  
 DBconnect.close()  
 return -1  
#endregion  
  
#region Add  
def AddMaxPlaceDef(add, ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("MaxPlaceDef", NameParkList, ParkName))  
 res = cursor.fetchall()  
 add += int(res[0][0])  
 cursor.execute("UPDATE {} SET MaxPlaceDef='{}' WHERE ParkName='{}'".format(NameParkList, str(add), ParkName))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при добавлении в MaxPlaceDef: " + e)  
 DBconnect.close()  
 return None  
  
  
def AddMaxPlaceVIP(add, ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("MaxPlaceVIP", NameParkList, ParkName))  
 res = cursor.fetchall()  
 add += int(res[0][0])  
 cursor.execute("UPDATE {} SET MaxPlaceVIP='{}' WHERE ParkName='{}'".format(NameParkList, str(add), ParkName))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при добавлении в MaxPlaceVIP: " + e)  
 DBconnect.close()  
 return None  
  
  
def AddFreePlaceDef(add, ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("FreePlaceDef", NameParkList, ParkName))  
 res = cursor.fetchall()  
 add += int(res[0][0])  
 cursor.execute("UPDATE {} SET FreePlaceDef='{}' WHERE ParkName='{}'".format(NameParkList, str(add), ParkName))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при добавлении в FreePlaceDef: " + e)  
 DBconnect.close()  
 return None  
  
  
def AddFreePlaceVIP(add, ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT {} FROM {} WHERE ParkName='{}'".format("FreePlaceVIP", NameParkList, ParkName))  
 res = cursor.fetchall()  
 add += int(res[0][0])  
 cursor.execute("UPDATE {} SET FreePlaceVIP='{}' WHERE ParkName='{}'".format(NameParkList, str(add), ParkName))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при добавлении в FreePlaceVIP: " + e)  
 DBconnect.close()  
 return None  
#endregion  
#endregion  
  
def Park(ParkName): # Есть ли уже навзание данной стоянки  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT ParkName FROM {}".format(NameParkList))  
 res = cursor.fetchall()  
  
 for i in range(0, len(res)):  
 if(res[i][0] == ParkName):  
 DBconnect.close()  
 return True  
  
 DBconnect.close()  
 return False  
 except Exception as e:  
 print("Ошибка при проверки стоянки: " + e)  
 DBconnect.close()  
 return None  
  
def NewPark(ParkName, MaxPlaceDef, MaxPlaceVIP):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName) # Подключение к базе (возмжоно нужно ватащить наружу)  
 cursor = DBconnect.cursor() # Место нахождение в таблице  
 cursor.execute("INSERT INTO {} (ParkName, MaxPlaceDef, FreePlaceDef, MaxPlaceVIP, FreePlaceVIP) VALUES ('{}', '{}','0', '{}', '0')".format(NameParkList, ParkName, str(MaxPlaceDef), str(MaxPlaceVIP)))  
 DBconnect.commit() # Синхронизирует изменения с баззой  
 DBconnect.close() # Закрывает открытую БД  
 except Exception as e:  
 print("Ошибка при занесении данных в базу: " + e)  
 DBconnect.close()  
 return None  
  
def DelPark(ParkName):  
 try:  
 DBconnect = sqlite3.connect(DataBaseName)  
 cursor = DBconnect.cursor()  
 cursor.execute("DELETE FROM {} WHERE ParkName='{}'".format(NameParkList, ParkName))  
 DBconnect.commit()  
 DBconnect.close()  
 except Exception as e:  
 print("Ошибка при удалении из базы: " + e)  
 DBconnect.close()  
 return None  
  
def ReturnTeble(Table):  
 DBconnect = sqlite3.connect(DataBaseName) # Подключение к базе (возмжоно нужно ватащить наружу)  
 cursor = DBconnect.cursor()  
 cursor.execute("SELECT \* FROM {}".format(Table))  
 res = cursor.fetchall()  
 DBconnect.close() # Закрывает открытую БД  
 return res

**Файл Parking.py**

from Stack import Stack  
from DataBase import\*  
from Car import Car  
from Tree import \*  
  
class Parking:  
 Name = None  
 MaxPlaceDef = 0  
 MaxPlaceVIP = 0  
 parkDef = Stack()  
 parkVIP = Stack()  
  
 def \_\_init\_\_(self, Name, PlaceDef, PlaceVip):  
 if(isinstance(Name, str) and isinstance(PlaceDef, int) and isinstance(PlaceVip, int)):  
 try:  
 self.Name = Name  
 self.MaxPlaceDef = PlaceDef  
 self.MaxPlaceVIP = PlaceVip  
 self.parkDef = Stack()  
 self.parkVIP = Stack()  
 if(not Park(self.Name)):  
 NewPark(self.Name, self.MaxPlaceDef, self.MaxPlaceVIP)  
 else:  
 Name = None  
 except Exception as e:  
 self.Name = None  
 print(e)  
 else:  
 Name = None  
  
 def CanAddToDef(self):  
 if(GetMaxPlaceDef(self.Name) - GetFreePlaceDef(self.Name) > 0):  
 return True  
 else:  
 return False  
  
 def CanAddToVIP(self):  
 if(GetMaxPlaceVIP(self.Name) - GetFreePlaceVIP(self.Name) > 0):  
 return True  
 else:  
 return False  
  
 def AddToDef(self, car):  
 if(self.CanAddToDef() and isinstance(car, Car)):  
 self.parkDef.push(car)  
  
 def AddToVIP(self, car):  
 if(self.CanAddToVIP() and isinstance(car, Car)):  
 self.parkVIP.push(car)  
  
 def BusyPlaceDef(self): # Возвращет True если на парковке 0 автомобилей и False, если на парковке есть хотя бы один автомобиль  
 if(GetFreePlaceDef(self.Name) == 0):  
 return True  
 else:  
 return False  
  
 def BusyPlaceVIP(self): # Возвращет True если на парковке 0 автомобилей  
 if(GetFreePlaceVIP(self.Name) == 0):  
 return True  
 else:  
 return False  
  
 def Dell(self): # Удаляет из БД  
 DelPark(self.Name)  
  
 def ParkCar(self, car, place = "DEF"):  
 AddToDataBase(car, None)  
 if(place == "DEF"):  
 self.AddToDef(car)  
 AddFreePlaceDef(1, self.Name)  
 else:  
 self.AddToVIP(car)  
 AddFreePlaceVIP(1, self.Name)  
  
 def PrintStackDef(self):  
 BufStack = Stack()  
 BufMirrorStack = Stack()  
  
 for i in range(self.parkDef.size()):  
 BufStack.push(self.parkDef.pop())  
 for i in range(BufStack.size()):  
 car = BufStack.pop()  
 BufMirrorStack.push(car)  
 self.parkDef.push(car)  
  
 list = []  
 for i in range(BufMirrorStack.size()):  
 list.append((BufMirrorStack.pop()).RegNum)  
 return list  
  
 def PrintStackVIP(self):  
 BufStack = Stack()  
 BufMirrorStack = Stack()  
  
 for i in range(self.parkVIP.size()):  
 BufStack.push(self.parkVIP.pop())  
 for i in range(BufStack.size()):  
 car = BufStack.pop()  
 BufMirrorStack.push(car)  
 self.parkVIP.push(car)  
  
 list = []  
 for i in range(BufMirrorStack.size()):  
 list.append((BufMirrorStack.pop()).RegNum)  
 return list  
  
 def CarOnPark(self, car):  
 BufStack = Stack()  
 BufMirrorStack = Stack()  
  
 for i in range(self.parkVIP.size()):  
 BufStack.push(self.parkVIP.pop())  
 for i in range(BufStack.size()):  
 car = BufStack.pop()  
 BufMirrorStack.push(car)  
 self.parkVIP.push(car)  
  
 for i in range(BufMirrorStack.size()):  
 if((BufMirrorStack.pop()).RegNum == car.RegNum):  
 return True  
  
 for i in range(self.parkDef.size()):  
 BufStack.push(self.parkDef.pop())  
 for i in range(BufStack.size()):  
 car = BufStack.pop()  
 BufMirrorStack.push(car)  
 self.parkDef.push(car)  
  
 for i in range(BufMirrorStack.size()):  
 if ((BufMirrorStack.pop()).RegNum == car.RegNum):  
 return True  
  
 return False  
  
 def DelCar(self, car):  
 BufStack = Stack()  
 CarHere = False  
 intMoved = None  
 for i in range(self.parkDef.size()):  
 newCar = self.parkDef.pop()  
 if(newCar.RegNum == car.RegNum): # Удаление машины  
 CarHere = True  
 intMoved = newCar.Moved  
 SetUnparking(None, car)  
 AddFreePlaceDef(-1, self.Name)  
 else:  
 newCar.Moved += 1  
 BufStack.push(newCar)  
 for i in range(BufStack.size()):  
 self.parkDef.push(BufStack.pop())  
  
 if(CarHere):  
 return intMoved  
  
 for i in range(self.parkVIP.size()):  
 newCar = self.parkVIP.pop()  
 if(newCar.RegNum == car.RegNum): # Удаление машины  
 CarHere = True  
 intMoved = newCar.Moved  
 SetUnparking(None, car)  
 AddFreePlaceVIP(-1, self.Name)  
 else:  
 BufStack.push(newCar)  
 for i in range(BufStack.size()):  
 self.parkVIP.push(BufStack.pop())  
  
 if (CarHere):  
 return intMoved  
  
 def ReturnCarListDef(self): # Возвращает представление парковки в виде массива  
 List = []  
 parkBuf = Stack()  
  
 for i in range(self.parkDef.size()): # Перегоняет машины в стек, что бы узнать какие машины на парковке  
 carBuf = self.parkDef.pop()  
 List.append(carBuf)  
 parkBuf.push(carBuf)  
 for i in range(parkBuf.size()): # Отрпавляет обратно в стек  
 self.parkDef.push(parkBuf.pop())  
 return List  
  
 def ReturnCarListVIP(self): # Возвращает представление парковки в виде массива  
 List = []  
 parkBuf = Stack()  
  
 for i in range(self.parkVIP.size()): # Перегоняет машины в стек, что бы узнать какие машины на парковке  
 carBuf = self.parkVIP.pop()  
 List.append(carBuf)  
 parkBuf.push(carBuf)  
 for i in range(parkBuf.size()): # Отрпавляет обратно в стек  
 self.parkVIP.push(parkBuf.pop())  
 return List  
  
  
 def SortTree(self): # Сортирует и VIP и обычную парковку  
 temp = [] # Сюда попадает отсортированные деревом ID автомобилей  
 ListDef = self.ReturnCarListDef()  
 TreeDef = None  
 for car in ListDef: # Перегоняет парковеу в дерево  
 TreeDef = insert(TreeDef, car.ID)  
 postorder(TreeDef, temp) # Получает от дерева отсортированный массив  
 self.parkDef = Stack() # Обнуляет стек  
  
 for carID in temp: # Отсортированный массив айдишников из дерева  
 for car in range(len(ListDef)): # Массив автомобилей созданый из стека  
 if(carID == ListDef[car].ID): # Если встречаеться айдишник, то машина попадает на парковку и удаляеться из ListDef для того что бы повторне не пробегать. Последним эллементом добавленным в стак будет корень дерева  
 self.parkDef.push(ListDef[car])  
 ListDef.pop(car)  
 break  
  
 temp = []  
 ListVIP = self.ReturnCarListVIP()  
 TreeVIP = None  
 for car in ListDef:  
 TreeVIP = insert(TreeVIP, car.ID)  
 postorder(TreeVIP, temp)  
 self.parkVIP = Stack()  
  
 for carID in temp:  
 for car in range(len(ListVIP)):  
 if (carID == ListVIP[car].ID):  
 self.parkVIP.push(ListVIP[car])  
 ListVIP.pop(car)  
 break

**Файл конструктора формы пользовательского интерфейса**

from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(716, 856)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.groupBox = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox.setGeometry(QtCore.QRect(10, 0, 691, 301))  
 self.groupBox.setObjectName("groupBox")  
 self.Table = QtWidgets.QTableWidget(self.groupBox)  
 self.Table.setEnabled(True)  
 self.Table.setGeometry(QtCore.QRect(20, 20, 651, 241))  
 self.Table.setColumnCount(4)  
 self.Table.setObjectName("Table")  
 self.Table.setRowCount(0)  
 self.PrintDB = QtWidgets.QPushButton(self.groupBox)  
 self.PrintDB.setGeometry(QtCore.QRect(20, 270, 141, 23))  
 self.PrintDB.setObjectName("PrintDB")  
 self.SelectDB = QtWidgets.QComboBox(self.groupBox)  
 self.SelectDB.setGeometry(QtCore.QRect(170, 270, 191, 22))  
 self.SelectDB.setObjectName("SelectDB")  
 self.SelectDB.addItem("")  
 self.SelectDB.addItem("")  
 self.SelectDB.addItem("")  
 self.groupBox\_2 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_2.setGeometry(QtCore.QRect(10, 310, 311, 271))  
 self.groupBox\_2.setObjectName("groupBox\_2")  
 self.groupBox\_3 = QtWidgets.QGroupBox(self.groupBox\_2)  
 self.groupBox\_3.setGeometry(QtCore.QRect(10, 20, 291, 151))  
 self.groupBox\_3.setObjectName("groupBox\_3")  
 self.TbParkName = QtWidgets.QLineEdit(self.groupBox\_3)  
 self.TbParkName.setGeometry(QtCore.QRect(10, 20, 113, 20))  
 self.TbParkName.setObjectName("TbParkName")  
 self.TbMaxDef = QtWidgets.QLineEdit(self.groupBox\_3)  
 self.TbMaxDef.setGeometry(QtCore.QRect(10, 50, 113, 20))  
 self.TbMaxDef.setObjectName("TbMaxDef")  
 self.TbMaxVIP = QtWidgets.QLineEdit(self.groupBox\_3)  
 self.TbMaxVIP.setGeometry(QtCore.QRect(10, 80, 113, 20))  
 self.TbMaxVIP.setObjectName("TbMaxVIP")  
 self.label = QtWidgets.QLabel(self.groupBox\_3)  
 self.label.setGeometry(QtCore.QRect(130, 20, 71, 16))  
 self.label.setObjectName("label")  
 self.label\_2 = QtWidgets.QLabel(self.groupBox\_3)  
 self.label\_2.setGeometry(QtCore.QRect(130, 50, 47, 13))  
 self.label\_2.setText("")  
 self.label\_2.setObjectName("label\_2")  
 self.label\_3 = QtWidgets.QLabel(self.groupBox\_3)  
 self.label\_3.setGeometry(QtCore.QRect(130, 50, 161, 16))  
 self.label\_3.setObjectName("label\_3")  
 self.label\_4 = QtWidgets.QLabel(self.groupBox\_3)  
 self.label\_4.setGeometry(QtCore.QRect(130, 80, 131, 16))  
 self.label\_4.setObjectName("label\_4")  
 self.BTCreatePark = QtWidgets.QPushButton(self.groupBox\_3)  
 self.BTCreatePark.setGeometry(QtCore.QRect(10, 110, 261, 23))  
 self.BTCreatePark.setObjectName("BTCreatePark")  
 self.groupBox\_4 = QtWidgets.QGroupBox(self.groupBox\_2)  
 self.groupBox\_4.setGeometry(QtCore.QRect(10, 180, 291, 81))  
 self.groupBox\_4.setObjectName("groupBox\_4")  
 self.SelectDelPark = QtWidgets.QComboBox(self.groupBox\_4)  
 self.SelectDelPark.setGeometry(QtCore.QRect(10, 20, 271, 22))  
 self.SelectDelPark.setObjectName("SelectDelPark")  
 self.BtDelPark = QtWidgets.QPushButton(self.groupBox\_4)  
 self.BtDelPark.setGeometry(QtCore.QRect(10, 50, 271, 23))  
 self.BtDelPark.setObjectName("BtDelPark")  
 self.groupBox\_5 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_5.setGeometry(QtCore.QRect(340, 310, 361, 151))  
 self.groupBox\_5.setObjectName("groupBox\_5")  
 self.SelectPark = QtWidgets.QComboBox(self.groupBox\_5)  
 self.SelectPark.setGeometry(QtCore.QRect(10, 20, 241, 22))  
 self.SelectPark.setObjectName("SelectPark")  
 self.TbCarNum = QtWidgets.QLineEdit(self.groupBox\_5)  
 self.TbCarNum.setGeometry(QtCore.QRect(10, 50, 141, 20))  
 self.TbCarNum.setObjectName("TbCarNum")  
 self.label\_5 = QtWidgets.QLabel(self.groupBox\_5)  
 self.label\_5.setGeometry(QtCore.QRect(260, 20, 101, 16))  
 self.label\_5.setObjectName("label\_5")  
 self.label\_6 = QtWidgets.QLabel(self.groupBox\_5)  
 self.label\_6.setGeometry(QtCore.QRect(160, 50, 141, 16))  
 self.label\_6.setObjectName("label\_6")  
 self.TbCarColor = QtWidgets.QLineEdit(self.groupBox\_5)  
 self.TbCarColor.setGeometry(QtCore.QRect(10, 80, 141, 20))  
 self.TbCarColor.setObjectName("TbCarColor")  
 self.label\_7 = QtWidgets.QLabel(self.groupBox\_5)  
 self.label\_7.setGeometry(QtCore.QRect(160, 80, 141, 16))  
 self.label\_7.setObjectName("label\_7")  
 self.BtParkCar = QtWidgets.QPushButton(self.groupBox\_5)  
 self.BtParkCar.setGeometry(QtCore.QRect(10, 110, 341, 23))  
 self.BtParkCar.setObjectName("BtParkCar")  
 self.groupBox\_6 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_6.setGeometry(QtCore.QRect(340, 470, 361, 111))  
 self.groupBox\_6.setObjectName("groupBox\_6")  
 self.TbCarNumLeave = QtWidgets.QLineEdit(self.groupBox\_6)  
 self.TbCarNumLeave.setGeometry(QtCore.QRect(10, 50, 141, 20))  
 self.TbCarNumLeave.setObjectName("TbCarNumLeave")  
 self.label\_9 = QtWidgets.QLabel(self.groupBox\_6)  
 self.label\_9.setGeometry(QtCore.QRect(160, 50, 141, 16))  
 self.label\_9.setObjectName("label\_9")  
 self.BtLeaveParkCar = QtWidgets.QPushButton(self.groupBox\_6)  
 self.BtLeaveParkCar.setGeometry(QtCore.QRect(10, 80, 341, 23))  
 self.BtLeaveParkCar.setObjectName("BtLeaveParkCar")  
 self.SelectParkLeave = QtWidgets.QComboBox(self.groupBox\_6)  
 self.SelectParkLeave.setGeometry(QtCore.QRect(10, 20, 241, 22))  
 self.SelectParkLeave.setObjectName("SelectParkLeave")  
 self.label\_8 = QtWidgets.QLabel(self.groupBox\_6)  
 self.label\_8.setGeometry(QtCore.QRect(260, 20, 101, 16))  
 self.label\_8.setObjectName("label\_8")  
 self.groupBox\_7 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_7.setGeometry(QtCore.QRect(10, 590, 261, 111))  
 self.groupBox\_7.setObjectName("groupBox\_7")  
 self.TbCarNumVIP = QtWidgets.QLineEdit(self.groupBox\_7)  
 self.TbCarNumVIP.setGeometry(QtCore.QRect(10, 20, 113, 20))  
 self.TbCarNumVIP.setObjectName("TbCarNumVIP")  
 self.label\_10 = QtWidgets.QLabel(self.groupBox\_7)  
 self.label\_10.setGeometry(QtCore.QRect(130, 20, 131, 16))  
 self.label\_10.setObjectName("label\_10")  
 self.BtAddVIP = QtWidgets.QPushButton(self.groupBox\_7)  
 self.BtAddVIP.setGeometry(QtCore.QRect(10, 50, 241, 23))  
 self.BtAddVIP.setObjectName("BtAddVIP")  
 self.BtDelVIP = QtWidgets.QPushButton(self.groupBox\_7)  
 self.BtDelVIP.setGeometry(QtCore.QRect(10, 80, 241, 23))  
 self.BtDelVIP.setObjectName("BtDelVIP")  
 self.groupBox\_8 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_8.setGeometry(QtCore.QRect(280, 590, 421, 81))  
 self.groupBox\_8.setObjectName("groupBox\_8")  
 self.SelectParkSort = QtWidgets.QComboBox(self.groupBox\_8)  
 self.SelectParkSort.setGeometry(QtCore.QRect(10, 20, 301, 22))  
 self.SelectParkSort.setObjectName("SelectParkSort")  
 self.label\_11 = QtWidgets.QLabel(self.groupBox\_8)  
 self.label\_11.setGeometry(QtCore.QRect(320, 20, 101, 16))  
 self.label\_11.setObjectName("label\_11")  
 self.TreeSort = QtWidgets.QPushButton(self.groupBox\_8)  
 self.TreeSort.setGeometry(QtCore.QRect(10, 50, 401, 23))  
 self.TreeSort.setObjectName("TreeSort")  
 self.groupBox\_9 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_9.setGeometry(QtCore.QRect(10, 710, 331, 91))  
 self.groupBox\_9.setObjectName("groupBox\_9")  
 self.SelectParkWrite = QtWidgets.QComboBox(self.groupBox\_9)  
 self.SelectParkWrite.setGeometry(QtCore.QRect(10, 20, 221, 22))  
 self.SelectParkWrite.setObjectName("SelectParkWrite")  
 self.label\_12 = QtWidgets.QLabel(self.groupBox\_9)  
 self.label\_12.setGeometry(QtCore.QRect(240, 20, 101, 16))  
 self.label\_12.setObjectName("label\_12")  
 self.PrintStack = QtWidgets.QPushButton(self.groupBox\_9)  
 self.PrintStack.setGeometry(QtCore.QRect(10, 50, 311, 31))  
 self.PrintStack.setObjectName("PrintStack")  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(MainWindow)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 716, 21))  
 self.menubar.setObjectName("menubar")  
 MainWindow.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName("statusbar")  
 MainWindow.setStatusBar(self.statusbar)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  
 self.groupBox.setTitle(\_translate("MainWindow", "Базы данных"))  
 self.PrintDB.setText(\_translate("MainWindow", "Вывести Базу Данных"))  
 self.SelectDB.setItemText(0, \_translate("MainWindow", "Вьезд выезд"))  
 self.SelectDB.setItemText(1, \_translate("MainWindow", "Парковочные места"))  
 self.SelectDB.setItemText(2, \_translate("MainWindow", "Лист VIP номеров"))  
 self.groupBox\_2.setTitle(\_translate("MainWindow", "Парковки"))  
 self.groupBox\_3.setTitle(\_translate("MainWindow", "Создать"))  
 self.label.setText(\_translate("MainWindow", "Имя парковки"))  
 self.label\_3.setText(\_translate("MainWindow", "Макс. кол-во обычных мест"))  
 self.label\_4.setText(\_translate("MainWindow", "Макс. кол-во VIP мест"))  
 self.BTCreatePark.setText(\_translate("MainWindow", "Создать парковку"))  
 self.groupBox\_4.setTitle(\_translate("MainWindow", "Удалить"))  
 self.BtDelPark.setText(\_translate("MainWindow", "Удалить пакровку"))  
 self.groupBox\_5.setTitle(\_translate("MainWindow", "Припарковать"))  
 self.label\_5.setText(\_translate("MainWindow", "Выбор парковки"))  
 self.label\_6.setText(\_translate("MainWindow", "Автомобильный номер"))  
 self.label\_7.setText(\_translate("MainWindow", "Цвет автомобиля"))  
 self.BtParkCar.setText(\_translate("MainWindow", "Припарковать автомобиль"))  
 self.groupBox\_6.setTitle(\_translate("MainWindow", "Выехать с парковки"))  
 self.label\_9.setText(\_translate("MainWindow", "Автомобильный номер"))  
 self.BtLeaveParkCar.setText(\_translate("MainWindow", "Выехать с парковки"))  
 self.label\_8.setText(\_translate("MainWindow", "Выбор парковки"))  
 self.groupBox\_7.setTitle(\_translate("MainWindow", "VIP"))  
 self.label\_10.setText(\_translate("MainWindow", "Автомобильный номер"))  
 self.BtAddVIP.setText(\_translate("MainWindow", "Добавить автомобиль в VIP лист"))  
 self.BtDelVIP.setText(\_translate("MainWindow", "Убрать автомобиль из VIP листа"))  
 self.groupBox\_8.setTitle(\_translate("MainWindow", "Сортировка парковки"))  
 self.label\_11.setText(\_translate("MainWindow", "Выбор парковки"))  
 self.TreeSort.setText(\_translate("MainWindow", "Сортировка двоичным деревом поиска"))  
 self.groupBox\_9.setTitle(\_translate("MainWindow", "Вывод инвормации"))  
 self.label\_12.setText(\_translate("MainWindow", "Выбор парковки"))  
 self.PrintStack.setText(\_translate("MainWindow", "Вывести информацию о парковке (выезд наверху)"))

**Основной файл программы Main.py**

import random  
import sys  
from PyQt5 import QtWidgets, QtCore  
from PyQt5.QtWidgets import QTableWidgetItem, QMessageBox  
  
from MainWindow import Ui\_MainWindow  
  
from DataBase import \*  
from Parking import Parking  
  
class Mywin(QtWidgets.QMainWindow):  
 ParkingsList = [] # Хранит себе все парковки  
 CountList = 0 # Кол-во парковок  
 def \_\_init\_\_(self): # Инициирует каждую кнопку  
 super(Mywin, self).\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
 self.ParkingsInit()  
 self.setWindowTitle("Py\_Kurs\_Work")  
 self.ui.PrintDB.clicked.connect(self.DrawOnTimeTable)  
 self.ui.Table.setColumnCount(4) # Кол-во столбов  
 self.ui.BTCreatePark.clicked.connect(self.AddParking)  
 self.ui.BtDelPark.clicked.connect(self.DellParking)  
 self.ui.BtAddVIP.clicked.connect(self.AddVIP)  
 self.ui.BtDelVIP.clicked.connect(self.DellVIP)  
 self.ui.BtParkCar.clicked.connect(self.ParkCar)  
 self.ui.PrintStack.clicked.connect(self.Print)  
 self.ui.BtLeaveParkCar.clicked.connect(self.OutFromParking)  
 self.ui.TreeSort.clicked.connect(self.ParkSort)  
 self.DrawOnTimeTable()  
  
 def ParkSort(self):  
 index = self.ui.SelectParkSort.currentIndex()  
 self.ParkingsList[index].SortTree()  
  
  
 def OutFromParking(self):  
 index = self.ui.SelectParkLeave.currentIndex()  
 CarNum = self.ui.TbCarNumLeave.text()  
 self.ui.TbCarNumLeave.clear()  
 CarNum = CarNum.strip()  
 car = Car("RandomColor", CarNum)  
 if(car.ID == None):  
 QMessageBox.critical(self, "Ошибка", "Введён не правильный автомобильный номер", QMessageBox.Ok)  
 else:  
 if(CarOnParing(car) and self.ParkingsList[index].CarOnPark(car)):  
 res = self.ParkingsList[index].DelCar(car)  
 self.DrawOnTimeTable() # Обнавялет БД  
 QMessageBox.information(self, "Парковочная информация", "Автомобиль с номером [{}] покинул парковку. \nАвтомобиль выезжал {} раз".format(car.RegNum, str(res)), QMessageBox.Ok)  
 else:  
 print(self.ParkingsList[index].CarOnPark(car))  
 QMessageBox.critical(self, "Ошибка", "Автомобиля нет на парковке", QMessageBox.Ok)  
  
 def Print(self): # Выводит месседж, что находиться на парковке  
 try:  
 # index = self.ui.SelectPark.currentIndex()  
 # name = self.ui.SelectPark.currentText()  
 index = self.ui.SelectParkWrite.currentIndex()  
 name = self.ui.SelectParkWrite.currentText()  
 list = self.ParkingsList[index].PrintStackDef()  
 ParkInfo = "Обычкая парковка:\n"  
 for string in list:  
 ParkInfo += string + "\n"  
 ParkInfo += "\n"  
 list = self.ParkingsList[index].PrintStackVIP()  
 ParkInfo += "VIP парковка:\n"  
 for string in list:  
 ParkInfo += string + "\n"  
 QMessageBox.information(self, "Парковка {}".format(name), ParkInfo, QMessageBox.Ok)  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", "Не получилось вывести парковку, нет парковок", QMessageBox.Ok)  
  
 def ParkCar(self): # Парковка автомобиля  
 index = self.ui.SelectPark.currentIndex()  
 ParkName = self.ui.SelectPark.currentText()  
 CarNum = self.ui.TbCarNum.text()  
 CarNum = CarNum.strip()  
 CarColor = self.ui.TbCarColor.text()  
 self.ui.TbCarNum.clear()  
 self.ui.TbCarColor.clear()  
 if(not (ParkName == "" or CarNum == "" or CarColor == "")): # Проверка заполнения полей  
 car = Car(CarColor, CarNum)  
 if(not car.ID == None): # Проверка на правильность автомобиля  
 if(CarOnParing(car)): # Припаркован ли автомобиль на какой-то из парковок  
 QMessageBox.critical(self, "Ошибка ", "Автомобиль ещё находиться на парковке",QMessageBox.Ok)  
 else:  
 if(VIP(car)): # Выбор типа парковки  
 if(self.ParkingsList[index].CanAddToVIP()):  
 self.ParkingsList[index].ParkCar(car, "VIP")  
 self.DrawOnTimeTable()  
 QMessageBox.information(self, "Парковочная информация","Автомобиль с номером [{}] припарковался.".format(car.RegNum), QMessageBox.Ok)  
 elif(self.ParkingsList[index].CanAddToDef()):  
 QMessageBox.critical(self, "Ошибка ", "На VIP парковке нет места. \nВы будете припаркованы на обычную парковку", QMessageBox.Ok)  
 self.ParkingsList[index].ParkCar(car)  
 self.DrawOnTimeTable()  
 QMessageBox.information(self, "Парковочная информация","Автомобиль с номером [{}] припарковался.".format(car.RegNum), QMessageBox.Ok)  
 else:  
 QMessageBox.critical(self, "Ошибка ", "На парковке нет места", QMessageBox.Ok)  
 else:  
 if(self.ParkingsList[index].CanAddToDef()):  
 self.ParkingsList[index].ParkCar(car)  
 self.DrawOnTimeTable()  
 QMessageBox.information(self, "Парковочная информация","Автомобиль с номером [{}] припарковался.".format(car.RegNum), QMessageBox.Ok)  
 else:  
 QMessageBox.critical(self, "Ошибка ", "На парковке нет места", QMessageBox.Ok)  
  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Автомобиль не может быть добавлен на парковку", QMessageBox.Ok)  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Поля ввода не могут оставаться пустыми", QMessageBox.Ok)  
  
 def AddParking(self): # Добавляет парковку в лист парковок  
 ParkName = self.ui.TbParkName.text()  
 MaxPlaceDef = self.ui.TbMaxDef.text()  
 MaxPlaceVIP = self.ui.TbMaxVIP.text()  
 if(not (ParkName == "" or MaxPlaceDef == "" or MaxPlaceVIP == "")):  
 self.ui.TbMaxVIP.setText(None)  
 self.ui.TbMaxDef.setText(None)  
 self.ui.TbParkName.setText(None)  
 if(not Park(ParkName)):  
 try:  
 self.ParkingsList.append(Parking(ParkName, int(MaxPlaceDef), int(MaxPlaceVIP)))  
 self.CountList -=-1  
 self.UpdateParkComboBox()  
 self.DrawOnTimeTable()  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка ", "Вы не можете добавить данную парковку", QMessageBox.Ok)  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Данная парковка уже существует", QMessageBox.Ok)  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Поля ввода не могут оставаться пустыми", QMessageBox.Ok)  
  
 def DellParking(self): # Удаление парковки из списка  
 text = self.ui.SelectDelPark.currentText()  
 if(not (text == "")):  
 if(self.CountList > 0): # Если нет парковок, то не пытаться их удалить  
 for i in range(self.CountList):  
 park = self.ParkingsList[i]  
 if(park.Name == text and (park.BusyPlaceDef() and park.BusyPlaceVIP())): # Если нашлось имя такое же как у нужной парковки и на ней нет машин, то можно удалять  
 self.ParkingsList[i].Dell() # Удаляет парковку из БД  
 self.ParkingsList.pop(i) # Удаляет эллемент по индексу  
 self.UpdateParkComboBox() # Обновляет все комбобоксы  
 self.CountList -= 1  
 self.DrawOnTimeTable()  
 break  
 elif(park.Name == text and (not park.BusyPlaceDef() or not park.BusyPlaceVIP())): # Не удалять парковку, пока на ней находиться автомобиль  
 QMessageBox.critical(self, "Ошибка ", "Вы не можете удалит парковку, ведь на ней есть автомобили", QMessageBox.Ok)  
 break  
 else:  
 QMessageBox.critical(self, "Ошибка", "Парковка для удаления не выбрана", QMessageBox.Ok)  
  
 def AddVIP(self): # Добавление машины в VIP лист  
 RegNum = self.ui.TbCarNumVIP.text()  
  
 if(not (RegNum == None or RegNum == "")):  
 car = Car("RandomColor", RegNum)  
 if(car.ID == None): # Класс машины делает ID = None если есть проблемы с автомобильным номером  
 QMessageBox.critical(self, "Ошибка ", "Был введён не правильный автомобильный номер \n[{}]".format(RegNum), QMessageBox.Ok)  
 else:  
 if(VIP(car)): # Проверка на наличие данного автомобиля в вип листе  
 QMessageBox.critical(self, "Ошибка ", "Машина уже находиться в VIP листе", QMessageBox.Ok)  
 else:  
 AddToVIP(car) # Добавление автомобиля в вип лист  
 self.DrawOnTimeTable()  
 self.ui.TbCarNumVIP.clear()  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Поле не может оставаться пустым", QMessageBox.Ok)  
  
 def DellVIP(self): # Удаление машины из VIP листа  
 RegNum = self.ui.TbCarNumVIP.text()  
 if (not (RegNum == None or RegNum == "")):  
 car = Car("RandomColor", RegNum)  
 if (car.ID == None):  
 QMessageBox.critical(self, "Ошибка ", "Был введён не правильный автомобильный номер \n[{}]".format(RegNum), QMessageBox.Ok)  
 else:  
 if(VIP(car)):  
 DellFromVIP(car)  
 self.DrawOnTimeTable()  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Машины нет в VIP листе", QMessageBox.Ok)  
 self.ui.TbCarNumVIP.clear()  
 else:  
 QMessageBox.critical(self, "Ошибка ", "Поле не может оставаться пустым", QMessageBox.Ok)  
  
 def UpdateParkComboBox(self): # Обнавляет все комбобоксы с парковками  
 self.ui.SelectDelPark.clear()  
 self.ui.SelectPark.clear()  
 self.ui.SelectParkLeave.clear()  
 self.ui.SelectParkSort.clear()  
 self.ui.SelectParkWrite.clear()  
  
 for park in self.ParkingsList:  
 self.ui.SelectDelPark.addItem(park.Name)  
 self.ui.SelectPark.addItem(park.Name)  
 self.ui.SelectParkLeave.addItem(park.Name)  
 self.ui.SelectParkSort.addItem(park.Name)  
 self.ui.SelectParkWrite.addItem(park.Name)  
  
 def DrawOnTimeTable(self): # Рисует выбранную пользователем БД  
 try:  
 buf = self.ui.SelectDB.currentIndex()  
 if(buf == 0):  
 data = ReturnTeble("CarParkingInfo") # Имя таблицы из которой будет рисоваться таблица  
 row = 0 # Строка  
 self.ui.Table.setRowCount(len(data)) # Кол-во строк  
 self.ui.Table.setColumnCount(4) # Кол-во столбов  
 self.ui.Table.setHorizontalHeaderLabels(("ID", "Номер", "Время въезда", "Время выезда")) # Имена столбов у таблицы  
 for tup in data: # Строка не поделённая на столбы  
 col = 0 # Столбец  
 for item in tup: # Значение каждой клетки в одной строке (Разных стобах)  
 cellinfo = QTableWidgetItem(str(item)) # Задаёт элемент  
 cellinfo.setFlags(QtCore.Qt.ItemIsSelectable | QtCore.Qt.ItemIsEnabled) # Только для чтения  
 self.ui.Table.setItem(row, col, cellinfo) # Вставляет элемент в определённую ячейку по строке/столбу  
 col -=-1  
 row -=-1  
  
 elif(buf == 1):  
 data = ReturnTeble("Places")  
 row = 0  
 self.ui.Table.setRowCount(len(data)) # Кол-во строк  
 self.ui.Table.setColumnCount(len(data[0])) # Кол-во столбов  
 self.ui.Table.setHorizontalHeaderLabels(("Имя парковки", "Макс. места на обычной","Занятое место на обычной", "Макс. места на VIP","Занятое место на VIP"))  
 for tup in data:  
 col = 0  
  
 for item in tup:  
 cellinfo = QTableWidgetItem(str(item))  
 cellinfo.setFlags(QtCore.Qt.ItemIsSelectable | QtCore.Qt.ItemIsEnabled) # Только для чтения  
 self.ui.Table.setItem(row, col, cellinfo)  
 col -=-1  
 row -=-1  
  
 elif(buf == 2):  
 data = ReturnTeble("VIP\_List")  
 row = 0  
 self.ui.Table.setRowCount(len(data)) # Кол-во строк  
 self.ui.Table.setColumnCount(len(data[0])) # Кол-во столбов  
 self.ui.Table.setHorizontalHeaderLabels(("ID", "Автомобильный номер"))  
 for tup in data:  
 col = 0  
 for item in tup:  
 cellinfo = QTableWidgetItem(str(item))  
 cellinfo.setFlags(QtCore.Qt.ItemIsSelectable | QtCore.Qt.ItemIsEnabled) # Только для чтения  
 self.ui.Table.setItem(row, col, cellinfo)  
 col -=-1  
 row -=-1  
  
 except Exception as e:  
 # self.ui.Table.setColumnCount(0) 2  
 # self.ui.Table.setRowCount(0) 1  
 # self.ui.Table.setItem(0, 0, QTableWidgetItem("Ошибка при выводе БД"))  
 # self.ui.Table.setItem(0, 1, QTableWidgetItem(str(e)))  
 QMessageBox.critical(self, "Ошибка", "Проблема с выводом базы данных.\nВозможно она пуста", QMessageBox.Ok)  
  
 def ParkingsInit(self): # Вытаскивает информацию о парковках из БД и отправляет её в ParkingsList  
 list = GetAllParkings()  
 for park in list:  
 self.ParkingsList.append(Parking(park[0], park[1], park[3]))  
 self.CountList -=-1  
 self.UpdateParkComboBox()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QtWidgets.QApplication(sys.argv)  
 win = Mywin()  
 win.show()  
 sys.exit(app.exec())