

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ рОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

|  |  |  |
| --- | --- | --- |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)**  **Департамент математического и компьютерного моделирования** | | |
|  | |
| «Персональный каталогизатор-систематизатор фотографий анималистического жанра на примере бёрдвотчинга» |
|  |

**КУРСОВАЯ РАБОТА**

по образовательной программе подготовки бакалавров

по направлению 02.03.01 «Математика и компьютерные науки»

профиль «Сквозные цифровые технологии»

|  |  |
| --- | --- |
| Работа защищена  с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  . | Студент группы № Б9119-02.03.01сцт  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панченко Н.К  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.  Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (должность, ученое звание)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (ФИО)  «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г. |

г. Владивосток

2022

**Оглавление**

|  |  |
| --- | --- |
| Аннотация  Введение  Основная часть  Заключение  Список литературы  Список использованных источников  Приложение | 3  4  5  21  22  22  24 |

**Аннотация**

В данной работе рассматривается задача реализации персональной базы данных для фотографа-анималиста. Полная реализация базы данных или каталогизатора должны включать следующий функционал:

* создание нового вида,
* загрузка фотографии или серий фотографий,
* привязка к фотографии заметок и геолокации,
* галерея по виду,
* выборку по критериям,
* галерея лучших снимков,
* возможность добавлять видеофрагменты и записи звуков.

В рамках курсовой работы требуется выполнить следующую часть работ: создать приложение для ПК, в котором будут следующие функции: добавление нового вида, общее описание вида, загрузка изображений, галерея по виду.

Решение выполнено с помощью интегрированной среды разработки PyCharm для языка программирования Python.

**Введение**

Бердвотчинг — это вид активного отдыха очень популярный в Америке и Западной Европе. Заключается в наблюдении за птицами, чаще всего с использованием бинокля или подзорной трубы, или фотографировании птиц, записи их голосов. Значительная часть научного орнитологического материала в мире собирается любителями-бёрдвотчерами. В тех уголках мира, куда бёрдвотчинг пришел в последние 25 лет, например в Азии и на постсоветском пространстве, в большей степени популярен фотобёрдвотчинг, то есть фотографирование птиц. Это связано в первую очередь с тем, что в это время широкой аудитории стали доступны как зеркальные камеры с телеобъективами, так и фотоаппараты-суперзумы. Каждый владелец такого фотоаппарата рано или поздно обязательно начинает делать фотографии птиц. Причин тому несколько. Во-первых, птицы заметны и хороши собой, но люди это замечают только взяв в руки бинокль или фотоаппарат. Во-вторых, птицы разнообразны, и у фотографа появляется желание находить и делать фотографии разных птиц. В-третьих, процесс фотоохоты сопровождается азартом и страстью, то есть, позволяет проявить древние инстинкты, но без крови. И в-четвертых, результат фотобёрдвотчинга — это прекрасные фотографии.

Цель создать персональное приложение для ПК, позволяющее фотографу-анималисту организовать каталог-систематизатор метаданных для личного архива анималистических фотографий. Например, редактируемый справочник видов, полевые заметки, географические привязки, дата съемки. Система должна предоставлять возможности поиска, фильтрации, систематизации фотографий.

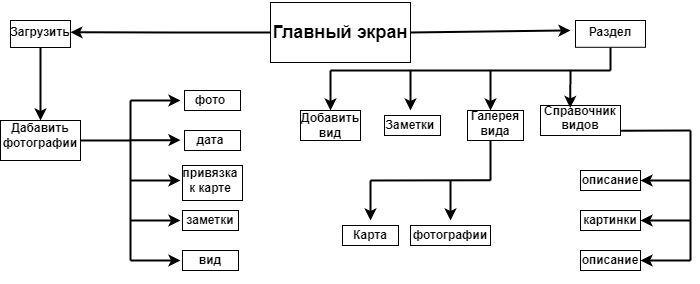
**Основная часть**

**Глава 1. Проект**

Каталогизатор должен включать следующий функционал: создание нового вида, загрузка фотографии или серий фотографий, привязка к фотографии заметок и геолокации, личные заметки по виду, галерея по виду, выборку по критериям, галерея лучших снимков, возможность добавлять видеофрагменты и записи звуков. Самое главное каталогизатор должен быть персональным.

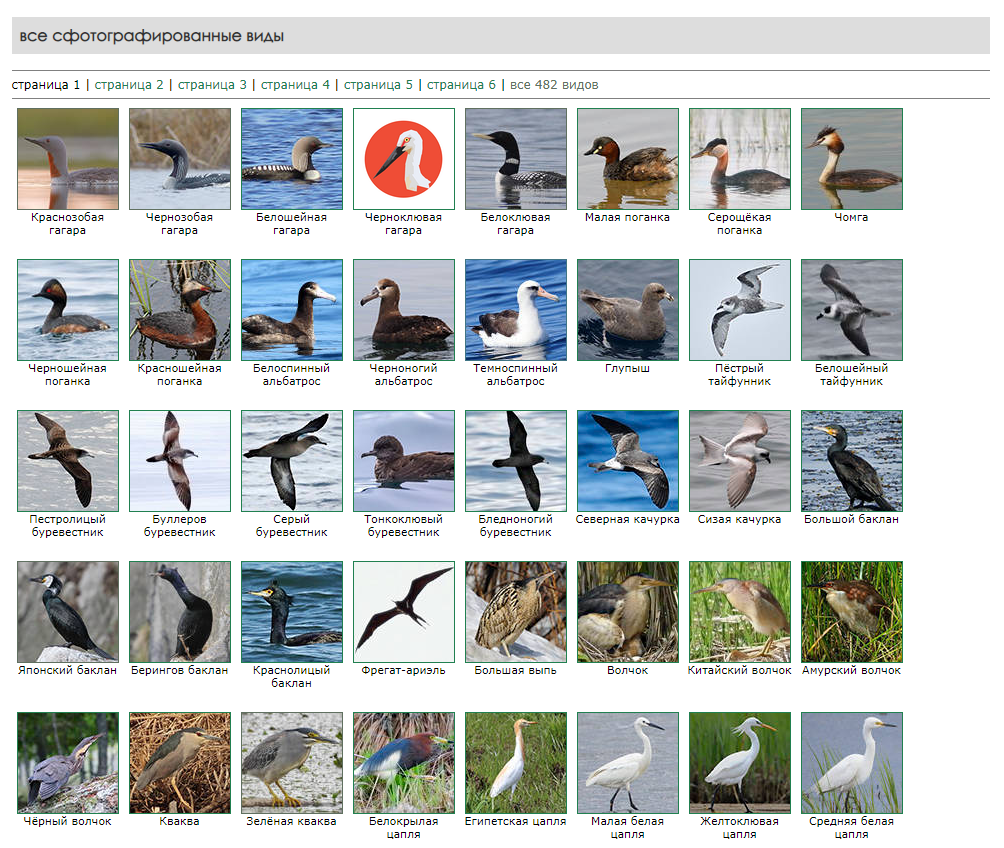
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Приложение** | **Персональный фото справочник** | **Добавление фотографий** | **Карта** | **Выборка по фильтрам** | **Добавление**  **вида** | **Лучшие**  **снимки** | **Видео**  **аудио** | **Описание вида** | **Не коммерческое** |
| Мое приложение | + | + | + | + | + | + | + | + | + |
| Adobe Lightroom | + | + | - | + | - | + | - | - | - |
| Adobe Bridge | + | + | - | + | - | - | - | - | - |
| Corel AfterShot Pro | + | - | - | + | - | + | + | - | - |
| ACDSee Photo Studio | + | - | - | + | - | + | + | - | - |
| ПДВ | - | + | + | + | + | + | + - | + | + |

ПДВ – сайт птицы Дальнего Востока функционально подходит, но не является персональным.



**Cхема приложения**

За основу общей организации интерфейса взят интерфейс интернет-ресурса «Птицы Дальнего Востока», части общего проекта бёрдвотчеров пост-советского пространства. Например:



**Рисунок 1: страница видов.**

Изображение выглядит как текст, снимок экрана, птица, водоплавающая птица

Автоматически созданное описание

**Рисунок 2: страница с общей информацией.**

Изображение выглядит как карта

Автоматически созданное описание

**Рисунок 3: страница галереи.**

**Глава 2.** Реализацияпроекта

**Требования к окружению**

Для разработки использовалась среда разработки PyCharm c использованием языка программирования Python и встроенными библиотеками, и модулями.

Системные требования для установки PyCharm:

* 64 – бит версия Windows 10, 8;
* 8ГБ оперативной памяти;
* 2.5ГБ свободного пространства на диске;
* Python 2.7, или Python 3.5 или новее.

PyCharm

Это кроссплатформенная среда разработки, которая совместима с Windows, macOS, Linux для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django(программа).

Для работы использовался Python-3.9, библиотеки Tkinter, PIL и модули SQlite3, IO.

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 в том плане, что всё является объектами. Стандартные библиотека включает большой объём полезных функций.

SQLite3

Модуль языка Python предоставляет интерфейс SQL, совместимый со спецификацией DB-API 2.0, описанной в PEP 249(спецификация API базы данных Python), и требует SQLite 3.7.15 или новее.

SQL

Декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

SQLite

SQLite — это библиотека C, которая предоставляет легкую дисковую базу данных, не требующую отдельного серверного процесса и позволяющую обращаться к базе данных с помощью нестандартного варианта языка запросов SQL. Некоторые приложения могут использовать SQLite для внутреннего хранения данных.

PIL(Python Imaging Library)

Библиотека языка Python (версии 2), предназначенная для работы с растровой графикой.

Возможности библиотеки:

* поддержка форматов BMP, EPS, GIF, JPEG, PDF, PNG, PNM, TIFF и некоторых других на чтение и запись;
* преобразование изображений из одного формата в другой;
* правка изображений (использование различных фильтров, масштабирование, рисование, матричные операции и т. д.).

IO

Модуль IO для выполнения операций ввода-вывода, связанные с файлами (например, чтение / запись файлов).

Tkinter

Это графическая библиотека, позволяющая создавать программы с оконным интерфейсом.

**Проект**

Методы и классы

Класс DB создаёт три таблицы для хранения информации с помощью запросов:

|  |
| --- |
| **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS birds (**  **id integer primary key, species text, date\_str, date\_int integer, place text, pic blob, notes text)''')**  **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS species (**  **species\_id integer primary key, name text, name\_lat text, name\_ing text, notes text, places text, biologi text, pic blob)''')**    **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS picture (**  **id integer primary key, name text, pic blob, pic\_min blob, pic\_mid blob)''')** |

**Листинг 1**

Класс ***Main*** запускает начальное окно программы.

Метод ***tk.Button*** – добавляет четыре кнопки действий(добавить категорию, добавить вид, добавить изображения, удалить). К кнопкам добавляются иконки.

Метод ***ttk.Treeview*** – создается таблица для отображения данных.

Метод ***tk.Scrollbar*** – добавляет полосу прокрутки для таблицы.

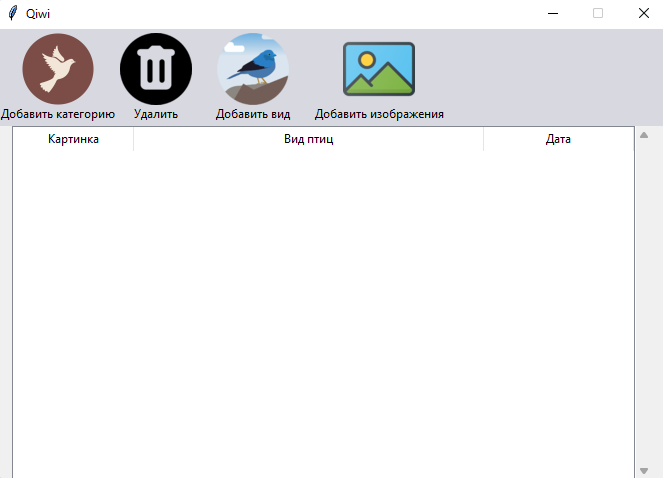
|  |
| --- |
| **tk.Button(toolbar, text='Добавить вид', bg='#d7d8e0', bd=0, image=self.species,**  **compound=tk.TOP, command=self.open\_dialog\_species)**  **ttk.Treeview(self, columns=('species', 'data'))**  **tk.Scrollbar(self, command=self.tree.yview)** |

**Листинг 2**

Метод ***view\_records*** – отображает данные в таблице.

|  |
| --- |
| **def view\_records(self):**  **self.db.c.execute('''SELECT pic, species, date\_str FROM birds''')**  **[self.tree.delete(i) for i in self.tree.get\_children()]**  **self.imglist = []**  **for record in self.db.c.fetchall():**  **img = ImageTk.PhotoImage(data=record[0])**  **self.tree.insert("", 'end', image=img, values=record[1:])**  **self.imglist.append(img)** |

**Листинг 3**



**Рисунок 4: основное окно приложения.**

Кнопка «Добавить вид» открывает окно, в котором можно создать новый вид для категорий.

Создаются поля для внесения информации методом ***ttk.Entry***.

Для больших полей с описание используется метод ***tk.Text***.

Для добавлений нужно нажать на кнопку «Добавить» вызовется метод, который считает данные с полей и добавит данные в таблицу в которой хранится информация с видами.

|  |
| --- |
| **label\_speceis = tk.Label(self, text='Название вида на русском:')**  **label\_speceis.place(x=15, y=20)**  **self.entry\_species = ttk.Entry(self, width=25)**  **self.entry\_species.place(x=200, y=20)**  **label\_notes = tk.Label(self, text='Описание:')**  **label\_notes.place(x=15, y=130)**  **self.text\_notes = tk.Text(self,width=43, height=8)**  **self.text\_notes.place(x=13, y=160)**  **btn\_submit = tk.Button(self, text='Добавить', width=20, font=('Helvetica', 11, 'bold'))**  **btn\_submit.place(x=110, y=648)**  **btn\_submit.bind('<Button-1>', lambda event: self.view.records\_for\_speces(self.entry\_species.get(),**  **self.entry\_species\_lat.get(),**  **self.entry\_species\_en.get(),**  **self.text\_notes.get(1.0, 'end-1c'),**  **self.text\_places.get(1.0, 'end-1c'),**  **self.text\_biologi.get(1.0, 'end-1c')))** |

**Листинг 4**

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 5: окно «Добавить вид».**

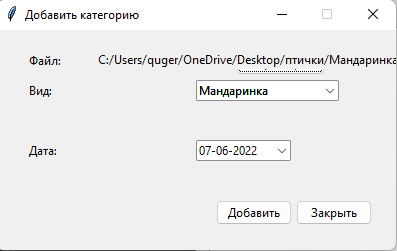
Кнопка «Добавить категорию» открывает окно, в котором вносятся данные для категории и добавляются в таблицу с категорией и отображается в таблице на главной странице.

В окне «Добавить категорию» расположены:

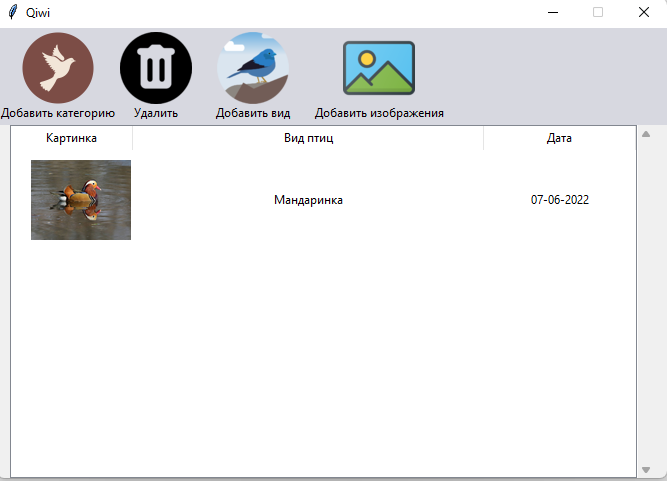
* кнопка «Выбрать файл» выбирается картинка для пред показа вида в категории;
* Выпадающий список, хранящий в себе добавленные виды;
* Поле даты для.
* Кнопка «Добавить» собирает данные и добавляет в таблицу для категорий, которые отображаются в таблице на главном окне.

|  |
| --- |
| **name\_values = self.get\_name()**  **self.entry\_species = ttk.Combobox(self, values=name\_values)**  **self.entry\_species.place(x=200, y=50)**  **btn\_file = ttk.Button(self, text='Выбрать файл', command=self.open\_file\_path)**  **btn\_file.place(x=240, y=20)**  **btn\_ok.bind('<Button-1>', lambda event: self.veiw.records(self.entry\_species.get(),**  **self.entry\_date.get(),**  **self.get\_timestamp\_from\_string(self.entry\_date.get()),**  **self.entry\_path.cget("text")))**  **def open\_file\_path(self):**  **filetypes = (('Image', '\*.jpg'), ('All files', '\*.\*'))**  **filename = fd.askopenfilename(title='Open a file', initialdir='/', filetypes=filetypes)**  **self.entry\_path = ttk.Label(self, text=filename)**  **self.entry\_path.place(x=100, y=20)** |

**Листинг 5**



**Рисунок 6: окно «Добавить категорию».**



**Рисунок 7: главное окно после добавления категории.**

Кнопка «Добавить изображение» открывает окно, в котором добавляются фотографии к виду.

В окне «Добавить категорию» расположены:

* Кнопка для выбора файла с изображением;
* Выпадающий список с выбором виду, к которому добавить изображение;
* Кнопка добавить, которая собирает выбранные данные и добавляет в таблицу с изображениями.

|  |
| --- |
| **name\_values = self.get\_name()**  **self.entry\_species = ttk.Combobox(self, values=name\_values,width=30)**  **self.entry\_species.place(x=150, y=50)**  **btn\_file = ttk.Button(self, text='Выбрать файл', command=self.open\_file\_path)**  **btn\_file.place(x=240, y=20)**  **btn\_ok = ttk.Button(self, text='Добавить')**  **btn\_ok.place(x=220, y=170)**  **btn\_ok.bind('<Button-1>', lambda event: self.veiw.records\_for\_picture(self.entry\_species.get(), self.entry\_path.cget("text")))** |

**Листинг 6.**

**Изображение выглядит как текст

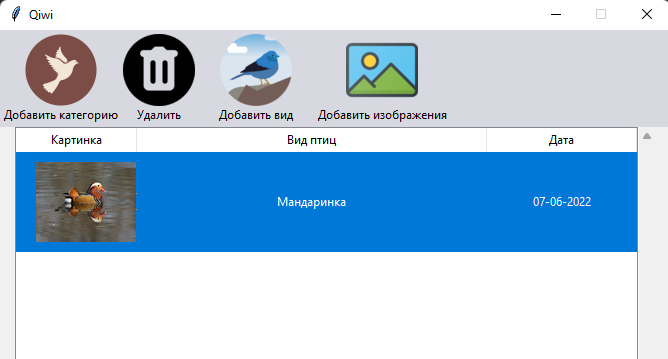
Автоматически созданное описание**

**Рисунок 8: окно «Добавить изображение».**

По двойному щелчку по выбранному элементу из таблицы на главном экране откроется окно, в котором собрана вся информация по виду и фотографии.

|  |
| --- |
| **def OnDoubleClick(self, event):**  **item = self.tree.identify('item', event.x, event.y)**  **self.open\_dialog\_view\_page()**  **def init\_veiw\_page(self):**  **self.title('Галерея')**  **self.geometry('1000x1000+400+1')**  **self.resizable(False, False)**  **data = self.view.get\_data\_species()**    **self.db.c.execute('''SELECT id, pic\_min, pic\_mid FROM picture WHERE name=?''', (data[0],))**  **e = self.get\_len\_data(data[3])**  **f = self.db.c.fetchall()**  **self.db.conn.commit()**  **self.a = self.get\_list\_pic\_mid(f)**  **self.imglist\_mid\_pic = []**  **for record in f:**  **img = ImageTk.PhotoImage(data=record[1])**  **self.imglist\_mid\_pic.append(img)**  **lbl\_name\_ru = ttk.Label(self, text=data[0],**  **font=("Arial", 28))**  **lbl\_name\_ru.place(x=40, y=10)**  **lbl\_name\_lat = ttk.Label(self, text=data[1],**  **font=("Arial", 16))**  **lbl\_name\_lat.place(x=40, y=60)**  **lbl\_name\_en = ttk.Label(self, text=data[2],**  **font=("Arial", 14))**  **lbl\_name\_en.place(x=40, y=90)**  **self.Artwork = tk.Label(self, image=self.a[0], background="black").place(x=40, y=120)**  **self.creat\_btn(f)**  **lbl\_notes = ttk.Label(self, text="Описание:", font=("Arial", 14))**  **lbl\_notes.place(x=40, y=610)**  **lbl\_notes\_text = ttk.Label(self, text=data[3], width= 130, font=("Arial", 10))**  **lbl\_notes\_text.place(x=40, y=640)**  **lbl\_notes\_text.bind('<Configure>', lambda e: lbl\_notes\_text.config(wraplength=lbl\_notes\_text.winfo\_width()))**  **lbl\_place = ttk.Label(self, text="Расположение:", font=("Arial", 14))**  **lbl\_place.place(x=40, y=730 + e)**  **lbl\_place\_text = ttk.Label(self, text=data[4], width=130, font=("Arial", 10))**  **lbl\_place\_text.place(x=40, y=760 + e)**  **lbl\_place\_text.bind('<Configure>', lambda e: lbl\_place\_text.config(wraplength=lbl\_place\_text.winfo\_width()))**  **lbl\_biologi = ttk.Label(self, text="Биология:",font=("Arial", 14))**  **lbl\_biologi.place(x=40, y=810 + e)**  **lbl\_biologi\_text = ttk.Label(self, text=data[5], width=130, font=("Arial", 10))**  **lbl\_biologi\_text.place(x=40, y=840 + e)**  **lbl\_biologi\_text.bind('<Configure>', lambda e: lbl\_biologi\_text.config(wraplength=lbl\_biologi\_text.winfo\_width()))** |

**Листинг 7**



**Рисунок 9: выбранный элемент**

**Рисунок 10: окно «Галерея»**

Если нажать на маленькое изображение оно отобразится в большем размере.

|  |  |
| --- | --- |
| Изображение выглядит как текст, птица, утка, водоплавающая птица  Автоматически созданное описание | Изображение выглядит как птица, другой, тот же  Автоматически созданное описание |

**Рисунок 11 Рисунок 12**

**Заключение**

В рамках курсовой работы мною был проведен анализ предметной области, поставленной задачи и имеющихся решений. Разработана архитектура системы и модель интерфейса. Реализована персональная база данных, как часть будущей системы каталогизатора.

В ходе работы были повышены навыки в программирование на Python и в использовании библиотеки tkinter и SQL запросов.

В дальнейшем разработка будет производиться с помощью другого графического интерфейса так как библиотека tkinter имеет ограниченный ресурс в сравнение с другими библиотеками.

**Список литературы**

1. Советов, Б. Я. Базы данных : учебник для среднего профессионального образования / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2020. — 420 с. — (Профессиональное образование). — ISBN 978-5-534-09324-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/453635 (дата обращения: 08.06.2022).
2. Сузи Р.А. Язык программирования Python : учебное пособие / Сузи Р.А.. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 350 c. — ISBN 978-5-4497-0705-5. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97589.html (дата обращения: 08.06.2022). — Режим доступа: для авторизир. Пользователей
3. Маккинли Уэс Python и анализ данных / Маккинли Уэс. — Саратов : Профобразование, 2019. — 482 c. — ISBN 978-5-4488-0046-7. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88752.html (дата обращения: 09.06.2022). — Режим доступа: для авторизир. пользователей

**Список интернет-источников**

<https://docs.python.org/3/library/sqlite3.html>

<https://docs.python.org/3/library/tkinter.html>

**Приложение**

|  |
| --- |
| **import sqlite3**  **import datetime**  **import tkinter as tk**  **import os**  **from tkinter import ttk**  **from tkcalendar import DateEntry, Calendar**  **from tkinter import filedialog as fd**  **from io import BytesIO**  **from PIL import ImageTk, Image**  **class Main(tk.Frame):**  **def \_\_init\_\_(self, root):**  **super().\_\_init\_\_(root)**  **self.init\_main()**  **self.db = db**  **self.view\_records()**  **def init\_main(self):**  **# СОЗДАНИЕ ПАНЕЛИ КНОПОК**  **toolbar = tk.Frame(bg='#d7d8e0', bd=2)**  **toolbar.pack(side=tk.TOP, fill=tk.X)**  **style = ttk.Style()**  **style.configure("Treeview", rowheight=100)**  **# СОЗДАНИЕ КНОПКИ С КАРТИНКОЙ ДОБАВИТЬ ИЗОБРАЖЕНИЕ**  **self.add\_img = tk.PhotoImage(file="icon/add\_icon.png")**  **btn\_open\_dialog = tk.Button(toolbar, text='Добавить категорию', command=self.open\_dialog, bg='#d7d8e0', bd=0,**  **compound=tk.TOP, image=self.add\_img)**  **btn\_open\_dialog.pack(side=tk.LEFT)**  **# СОЗДАНИЕ КНОПКИ С КАРТИНКОЙ УДАЛИТЬ**  **self.delete\_img = tk.PhotoImage(file='icon/trash.png')**  **btn\_delete = tk.Button(toolbar, text='Удалить', bg='#d7d8e0', bd=0, image=self.delete\_img,**  **compound=tk.TOP, command=self.delete\_records)**  **btn\_delete.pack(side=tk.LEFT)**  **# СОЗДАНИЕ КНОПКИ ДОБАВИТЬ ВИД**  **self.species = tk.PhotoImage(file='icon/sp.png')**  **btn\_add\_species = tk.Button(toolbar, text='Добавить вид', bg='#d7d8e0', bd=0, image=self.species,**  **compound=tk.TOP, command=self.open\_dialog\_species)**  **btn\_add\_species.pack(side=tk.LEFT, padx=20)**  **self.images = tk.PhotoImage(file='icon/image.png')**  **images = tk.Button(toolbar, text='Добавить изображения', bg='#d7d8e0', bd=0, image=self.images,**  **compound=tk.TOP, command=self.open\_dialog\_add\_images\_page)**  **images.pack(side=tk.LEFT)**  **# СОЗДАНИЕ ТАБЛИЦЫ С КОЛОНКАМИ**  **self.tree = ttk.Treeview(self, columns=('species', 'data'))**  **self.tree.column("species", width=350, anchor=tk.CENTER)**  **self.tree.column("data", width=150, anchor=tk.CENTER)**  **self.tree.column("#0", width=120)**  **# ЗАГОЛОВКИ**  **self.tree.heading("species", text='Вид птиц')**  **self.tree.heading("data", text='Дата')**  **self.tree.heading("#0", text="Картинка")**  **# УПАКОВКА ТАБЛИЦЫ**  **self.tree.pack(side=tk.LEFT)**  **self.tree.bind("<Double-1>", self.OnDoubleClick)**  **scroll = tk.Scrollbar(self, command=self.tree.yview)**  **scroll.pack(side=tk.RIGHT, fill=tk.Y)**  **self.tree.configure(yscrollcommand=scroll.set)**  **# УДАЛЕНИЕ ИЗ ТАБЛИЦЫ И БД**  **def delete\_records(self):**  **for selection\_item in self.tree.selection():**  **self.db.c.execute('''DELETE FROM birds WHERE species=?''', (self.tree.set(selection\_item, '#1'),))**  **self.db.conn.commit()**  **self.view\_records()**  **def OnDoubleClick(self, event):**  **item = self.tree.identify('item', event.x, event.y)**  **self.open\_dialog\_view\_page()**  **def get\_data\_species(self):**  **self.db.c.execute('''SELECT name, name\_lat, name\_ing, notes, places , biologi FROM species WHERE name=?''',**  **(self.tree.set(self.tree.selection()[0], '#1'),))**  **data\_kek = []**  **for row in self.db.c.fetchall():**  **data\_kek.append(row)**  **lst\_data = list(data\_kek[0])**  **return lst\_data**  **def records\_for\_speces(self, name, name\_lat, name\_ing, notes, places , biologi):**  **self.db.insert\_speceis(name, name\_lat, name\_ing, notes, places , biologi)**  **# МЕТОД ДЛЯ ЗАПИСИ В БД И ОТОПБРАЖЕНИЯ В ТАБЛИЦЕ**  **def records(self, species, date\_str, date\_int, pic):**  **self.db.insert\_data(species, date\_str, date\_int, pic)**  **self.view\_records()**  **def records\_for\_picture(self, name, pic):**  **self.db.insert\_picture(name, pic)**  **# МЕТОД ОТОБРАЖЕНИЯ ДАННЫХ**  **def view\_records(self):**  **self.db.c.execute('''SELECT pic, species, date\_str FROM birds''')**  **[self.tree.delete(i) for i in self.tree.get\_children()]**  **self.imglist = []**  **for record in self.db.c.fetchall():**  **img = ImageTk.PhotoImage(data=record[0])**  **self.tree.insert("", 'end', image=img, values=record[1:])**  **self.imglist.append(img)**  **def open\_dialog(self):**  **Child()**  **def open\_dialog\_species(self):**  **Speceis()**  **def open\_dialog\_view\_page(self):**  **VeiwPage()**  **def open\_dialog\_add\_images\_page(self):**  **Picture()**  **class Picture(tk.Toplevel):**  **def \_\_init\_\_(self):**  **super().\_\_init\_\_(root)**  **self.veiw = app**  **self.db = db**  **self.init\_child()**  **def init\_child(self):**  **# СОЗДАНИЕ ДИАЛОГОВОГО ОКНА**  **self.title('Добавить изображение')**  **self.geometry('400x220+400+300')**  **self.resizable(False, False)**  **# СОЗДАНИЕ ЛЕЙБЛОВ ДЛЯ ПОДПИСИ**  **label\_path = tk.Label(self, text='Файл:')**  **label\_path.place(x=30, y=20)**  **label\_species = tk.Label(self, text='Вид:')**  **label\_species.place(x=30, y=50)**  **name\_values = self.get\_name()**  **self.entry\_species = ttk.Combobox(self, values=name\_values,width=30)**  **self.entry\_species.place(x=150, y=50)**  **btn\_file = ttk.Button(self, text='Выбрать файл', command=self.open\_file\_path)**  **btn\_file.place(x=240, y=20)**  **btn\_ok = ttk.Button(self, text='Добавить')**  **btn\_ok.place(x=220, y=170)**  **btn\_ok.bind('<Button-1>', lambda event: self.veiw.records\_for\_picture(self.entry\_species.get(),**  **self.entry\_path.cget("text")))**  **# ПРИОРИТЕТ ДЛЯ 2-ГО ОКНА**  **self.grab\_set()**  **self.focus\_set()**  **def get\_name(self):**  **self.db.c.execute('''SELECT name FROM species''')**  **data\_name = []**  **for row in self.db.c.fetchall():**  **data\_name.append(row[0])**  **return data\_name**  **def open\_file\_path(self):**  **filetypes = (('Image', '\*.jpg'), ('All files', '\*.\*'))**  **filename = fd.askopenfilename(title='Open a file', initialdir='/', filetypes=filetypes)**  **os.path.basename(filename)**  **self.entry\_path = ttk.Label(self, text=filename)**  **self.entry\_path.place(x=100, y=20)**  **class Child(tk.Toplevel):**  **def \_\_init\_\_(self):**  **super().\_\_init\_\_(root)**  **self.veiw = app**  **self.db = db**  **self.init\_child()**  **def init\_child(self):**  **# СОЗДАНИЕ ДИАЛОГОВОГО ОКНА**  **self.title('Добавить изображение')**  **self.geometry('400x220+400+300')**  **self.resizable(False, False)**  **# СОЗДАНИЕ ЛЕЙБЛОВ ДЛЯ ПОДПИСИ**  **label\_path = tk.Label(self, text='Файл:')**  **label\_path.place(x=30, y=20)**  **label\_species = tk.Label(self, text='Вид:')**  **label\_species.place(x=30, y=50)**  **# label\_place = tk.Label(self, text='Место съемки:')**  **# label\_place.place(x=30, y=80)**  **label\_select = tk.Label(self, text='Дата:')**  **label\_select.place(x=30, y=110)**  **# ПОЛЯ ДЛЯ ВВОДА ДАННЫХ**  **self.entry\_path = ttk.Label(self, text='тут путь к файлу')**  **self.entry\_path.place(x=100, y=20)**  **name\_values = self.get\_name()**  **self.entry\_species = ttk.Combobox(self, values=name\_values)**  **self.entry\_species.place(x=200, y=50)**  **# self.entry\_place = ttk.Entry(self)**  **# self.entry\_place.place(x=200, y=80)**  **self.entry\_date = DateEntry(self, date\_pattern='dd-mm-YYYY')**  **self.entry\_date.place(x=200, y=110)**  **#КНОПОКИ ДЛЯ ДИАЛОГОВОГО ОКНА**  **btn\_file = ttk.Button(self, text='Выбрать файл', command=self.open\_file\_path)**  **btn\_file.place(x=240, y=20)**  **btn\_cancel = ttk.Button(self, text='Закрыть', command=self.destroy)**  **btn\_cancel.place(x=300, y=170)**  **btn\_ok = ttk.Button(self, text='Добавить')**  **btn\_ok.place(x=220, y=170)**  **btn\_ok.bind('<Button-1>', lambda event: self.veiw.records(self.entry\_species.get(),**  **self.entry\_date.get(),**  **self.get\_timestamp\_from\_string(self.entry\_date.get()),**  **self.entry\_path.cget("text")))**  **# ПРИОРИТЕТ ДЛЯ 2-ГО ОКНА**  **self.grab\_set()**  **self.focus\_set()**  **def get\_name(self):**  **self.db.c.execute('''SELECT name FROM species''')**  **data\_name = []**  **for row in self.db.c.fetchall():**  **data\_name.append(row[0])**  **return data\_name**  **# ПОЛУЧЕНИЕ ДАТЫ В ВИДЕ INTEGER**  **def get\_timestamp(self, y, m, d):**  **return int(datetime.datetime.timestamp(datetime.datetime(y, m, d)))**  **# ПОЛУЧЕНИЕ ДАТЫ ИЗ СТРОКИ В INTEGER**  **def get\_timestamp\_from\_string(self, s):**  **t = s.split('-')**  **return self.get\_timestamp(int(t[2]), int(t[1]), int(t[0]))**  **# ОТКРЫВАЕТ ОКНО**  **def open\_file\_path(self):**  **filetypes = (('Image', '\*.jpg'), ('All files', '\*.\*'))**  **filename = fd.askopenfilename(title='Open a file', initialdir='/', filetypes=filetypes)**  **self.entry\_path = ttk.Label(self, text=filename)**  **self.entry\_path.place(x=100, y=20)**  **class VeiwPage(tk.Toplevel):**  **def \_\_init\_\_(self):**  **super().\_\_init\_\_()**  **self.view = app**  **self.db = db**  **self.init\_veiw\_page()**  **def init\_veiw\_page(self):**  **self.title('Галерея')**  **self.geometry('1000x1000+400+1')**  **self.resizable(False, False)**  **data = self.view.get\_data\_species()**  **# print(data[0])**  **self.db.c.execute('''SELECT id, pic\_min, pic\_mid FROM picture WHERE name=?''', (data[0],))**  **e = self.get\_len\_data(data[3])**  **f = self.db.c.fetchall()**  **self.db.conn.commit()**  **self.a = self.get\_list\_pic\_mid(f)**  **self.imglist\_mid\_pic = []**  **for record in f:**  **img = ImageTk.PhotoImage(data=record[1])**  **self.imglist\_mid\_pic.append(img)**  **lbl\_name\_ru = ttk.Label(self, text=data[0],**  **font=("Arial", 28))**  **lbl\_name\_ru.place(x=40, y=10)**  **lbl\_name\_lat = ttk.Label(self, text=data[1],**  **font=("Arial", 16))**  **lbl\_name\_lat.place(x=40, y=60)**  **lbl\_name\_en = ttk.Label(self, text=data[2],**  **font=("Arial", 14))**  **lbl\_name\_en.place(x=40, y=90)**  **self.Artwork = tk.Label(self, image=self.a[0], background="black").place(x=40, y=120)**  **self.creat\_btn(f)**  **lbl\_notes = ttk.Label(self, text="Описание:", font=("Arial", 14))**  **lbl\_notes.place(x=40, y=610)**  **lbl\_notes\_text = ttk.Label(self, text=data[3], width= 130, font=("Arial", 10))**  **lbl\_notes\_text.place(x=40, y=640)**  **lbl\_notes\_text.bind('<Configure>', lambda e: lbl\_notes\_text.config(wraplength=lbl\_notes\_text.winfo\_width()))**  **lbl\_place = ttk.Label(self, text="Расположение:", font=("Arial", 14))**  **lbl\_place.place(x=40, y=730 + e)**  **lbl\_place\_text = ttk.Label(self, text=data[4], width=130, font=("Arial", 10))**  **lbl\_place\_text.place(x=40, y=760 + e)**  **lbl\_place\_text.bind('<Configure>', lambda e: lbl\_place\_text.config(wraplength=lbl\_place\_text.winfo\_width()))**  **lbl\_biologi = ttk.Label(self, text="Биология:",font=("Arial", 14))**  **lbl\_biologi.place(x=40, y=810 + e)**  **lbl\_biologi\_text = ttk.Label(self, text=data[5], width=130, font=("Arial", 10))**  **lbl\_biologi\_text.place(x=40, y=840 + e)**  **lbl\_biologi\_text.bind('<Configure>', lambda e: lbl\_biologi\_text.config(wraplength=lbl\_biologi\_text.winfo\_width()))**  **if len(data[4]) == 0:**  **lbl\_place.destroy()**  **if len(data[5]) == 0:**  **lbl\_biologi.destroy()**  **print(len(data[4]))**  **def creat\_btn(self, fetch):**  **self.list\_pic = []**  **r = 0**  **for record in fetch:**  **img = ImageTk.PhotoImage(data=record[1])**  **self.list\_pic.append(img)**  **list\_btn = []**  **bt = tk.Button(self, compound=tk.TOP, font=('Arial', 10), bd=0)**  **bt.bind("<Button-1>", lambda event: tk.Label(self, image=self.a[0], background="black").place(x=40, y=120))**  **bt.place(x=700, y=120)**  **list\_btn.append(bt)**  **bt1 = tk.Button(self, compound=tk.TOP, font=('Arial', 10),bd=0)**  **bt1.bind("<Button-1>", lambda event: tk.Label(self, image=self.a[1], background="black").place(x=40, y=120))**  **bt1.place(x=700, y=220)**  **list\_btn.append(bt1)**  **bt2 = tk.Button(self, compound=tk.TOP, font=('Arial', 10),bd=0)**  **bt2.bind("<Button-1>", lambda event: tk.Label(self, image=self.a[2], background="black").place(x=40, y=120))**  **bt2.place(x=700, y=320)**  **list\_btn.append(bt2)**  **bt3 = tk.Button(self, compound=tk.TOP, font=('Arial', 10),bd=0)**  **bt3.bind("<Button-1>", lambda event: tk.Label(self, image=self.a[3], background="black").place(x=40, y=120))**  **bt3.place(x=700, y=420)**  **list\_btn.append(bt3)**  **for i in range(len(self.list\_pic)):**  **list\_btn[i] = list\_btn[i].configure(image=self.list\_pic[i])**  **def get\_list\_pic\_min(self, fetch):**  **self.list\_pic = []**  **r = 0**  **for record in fetch:**  **img = ImageTk.PhotoImage(data=record[1])**  **self.list\_pic.append(img)**  **b = []**  **for i in range(len(self.list\_pic)):**  **bt = tk.Button(self,compound=tk.TOP, image=self.list\_pic[i], font=('Arial', 10))**  **bt.bind("<Button-1>", lambda event: tk.Label(self, image=self.a[i], background="black").place(x=40, y=120))**  **bt.place(x=700, y=120 + r)**  **b.append(bt)**  **r += 120**  **for i in range(len(b)):**  **b[i] = b[i].configure(text=str(i))**  **def get\_list\_pic\_mid(self, fetch):**  **self.list\_pic = []**  **r = 0**  **for record in fetch:**  **img = ImageTk.PhotoImage(data=record[2])**  **self.list\_pic.append(img)**  **return self.list\_pic**  **def get\_len\_data(self, data):**  **e = 0**  **if len(data) > 450:**  **e = 40**  **return e**  **class Speceis(tk.Toplevel):**  **def \_\_init\_\_(self):**  **super().\_\_init\_\_()**  **self.init\_speceis()**  **self.view = app**  **def init\_speceis(self):**  **self.title('Добавить вид')**  **self.geometry('400x690+800+200')**  **self.resizable(False, False)**  **label\_speceis = tk.Label(self, text='Название вида на русском:')**  **label\_speceis.place(x=15, y=20)**  **self.entry\_species = ttk.Entry(self, width=25)**  **self.entry\_species.place(x=200, y=20)**  **label\_speceis\_lat = tk.Label(self, text='Название вида на латинице:')**  **label\_speceis\_lat.place(x=15, y=60)**  **self.entry\_species\_lat = ttk.Entry(self, width=25)**  **self.entry\_species\_lat.place(x=200, y=60)**  **label\_speceis\_en = tk.Label(self, text='Название вида на английском:')**  **label\_speceis\_en.place(x=15, y=100)**  **self.entry\_species\_en = ttk.Entry(self, width=25)**  **self.entry\_species\_en.place(x=200, y=100)**  **label\_notes = tk.Label(self, text='Описание:')**  **label\_notes.place(x=15, y=130)**  **self.text\_notes = tk.Text(self,width=43, height=8)**  **self.text\_notes.place(x=13, y=160)**  **label\_places = tk.Label(self, text='Распространение:')**  **label\_places.place(x=15, y=300)**  **self.text\_places = tk.Text(self, width=43, height=8)**  **self.text\_places.place(x=15, y=330)**  **label\_biologi = tk.Label(self, text='Биология:')**  **label\_biologi.place(x=15, y=470)**  **self.text\_biologi= tk.Text(self, width=43, height=8)**  **self.text\_biologi.place(x=15, y=500)**  **btn\_submit = tk.Button(self, text='Добавить', width=20, font=('Helvetica', 11, 'bold'))**  **btn\_submit.place(x=110, y=648)**  **btn\_submit.bind('<Button-1>', lambda event: self.view.records\_for\_speces(self.entry\_species.get(),**  **self.entry\_species\_lat.get(),**  **self.entry\_species\_en.get(),**  **self.text\_notes.get(1.0, 'end-1c'),**  **self.text\_places.get(1.0, 'end-1c'),**  **self.text\_biologi.get(1.0, 'end-1c')))**  **self.grab\_set()**  **self.focus\_set()**  **class DB:**  **# СОЗДАНИЕ БД**  **def \_\_init\_\_(self):**  **self.conn = sqlite3.connect('db/database.db')**  **self.c = self.conn.cursor()**  **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS birds (**  **id integer primary key, species text, date\_str, date\_int integer, place text, pic blob, notes text)''')**  **self.conn.commit()**  **# СОЗДАНИЕ БД**  **self.conn = sqlite3.connect('db/database.db')**  **self.c = self.conn.cursor()**  **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS species (**  **species\_id integer primary key, name text, name\_lat text, name\_ing text, notes text, places text, biologi text, pic blob)''')**  **self.conn.commit()**  **self.conn = sqlite3.connect('db/database.db')**  **self.c = self.conn.cursor()**  **self.c.execute(**  **'''CREATE TABLE IF NOT EXISTS picture (**  **id integer primary key, name text, pic blob, pic\_min blob, pic\_mid blob)''')**  **self.conn.commit()**  **# ВСТАВКА ДАННЫХ ИЗ ФОРМЫ**  **def insert\_data(self, species, date\_str, date\_int, pic):**  **img = Image.open(pic)**  **img = img.resize((100, 80), Image.ANTIALIAS)**  **with BytesIO() as f:**  **img.save(f, 'PNG')**  **self.fob = f.getvalue()**  **self.c.execute('''INSERT INTO birds(species, date\_str, date\_int, pic) VALUES (?, ?, ?, ?)''',**  **(species, date\_str, date\_int, self.fob))**  **self.conn.commit()**  **def insert\_speceis(self, name, name\_lat, name\_ing, notes, places , biologi):**  **self.c.execute('''INSERT INTO species (name, name\_lat, name\_ing, notes, places , biologi) VALUES (?, ?, ?, ?, ?, ?)''',**  **(name, name\_lat, name\_ing, notes, places , biologi))**  **self.conn.commit()**  **def insert\_picture(self,name, pic):**  **img\_norm = Image.open(pic)**  **img\_min = Image.open(pic)**  **img\_min = img\_min.resize((120, 90))**  **img\_mid= Image.open(pic)**  **img\_mid = img\_mid.resize((640, 480))**  **with BytesIO() as f:**  **img\_norm.save(f, 'PNG')**  **self.fob = f.getvalue()**  **with BytesIO() as f:**  **img\_min.save(f, 'PNG')**  **self.fob\_min = f.getvalue()**  **with BytesIO() as f:**  **img\_mid.save(f, 'PNG')**  **self.fob\_mid = f.getvalue()**  **print(self.fob)**  **self.c.execute('''INSERT INTO picture(name, pic,pic\_min, pic\_mid) VALUES (?, ?, ?, ?)''',**  **(name, self.fob, self.fob\_min, self.fob\_mid))**  **self.conn.commit()**  **print("add")**  **if \_\_name\_\_ == "\_\_main\_\_":**  **root = tk.Tk()**  **db = DB()**  **app = Main(root)**  **app.pack()**  **root.title("Qiwi ")**  **root.geometry("670x450+600+200")**  **root.resizable(False, False)**  **root.mainloop()** |