

«Todolist» на Python с использованием SQLite

Акентьев Данила Денисович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс разработки приложения для управления задачами (Todolist) на языке программирования Python с использованием базы данных SQLite для хранения данных. Основное внимание уделено функциональности приложения, включая авторизацию пользователей, создание, редактирование, удаление задач и изменение их статуса. Рассматривается архитектура приложения, выбор инструментов и технологий, а также методы работы с графическим интерфейсом и базой данных. Реализованный интерфейс на базе библиотеки Tkinter предоставляет пользователю удобный и интуитивно понятный способ взаимодействия с приложением. В работе приведены результаты тестирования и обсуждаются перспективы развития проекта.

Ключевые слова: Python, SQLite, Tkinter, приложение для задач, авторизация, регистрация, базы данных, разработка приложений, интерфейс пользователя, статус задач.

"Todolist" in Python using SQLite

Akentev Danila Denisovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses the process of developing a task management application (Todolist) in the Python programming language using an SQLite database for data storage. The main focus is on the functionality of the application, including user authorization, creation, editing, deletion of tasks and changing their status. The architecture of the application, the choice of tools and technologies, as well as methods of working with the graphical interface and database are considered. The implemented interface based on the Tkinter library provides the user with a convenient and intuitive way to interact with the application. The paper presents the test results and discusses the prospects for the development of the project.

Keywords: Python, SQLite, Tkinter, task application, authorization, registration, databases, application development, user interface, task status.

1 Введение

1.1 Актуальность

С каждым годом количество информации, с которой работает человек, растет. Это приводит к необходимости использования различных инструментов для эффективного управления задачами и времени. Разработка приложений для учета задач на локальных устройствах предоставляет удобство и независимость от облачных решений. В связи с этим возникает потребность в простых и доступных решениях для создания таких приложений, особенно для пользователей с базовыми навыками программирования. Работа с базой данных SQLite позволяет обеспечить удобное хранение и управление задачами без необходимости настройки сложных серверных инфраструктур.

1.2 Обзор исследований

Ранее вопросы разработки приложений с графическим пользовательским интерфейсом (GUI) с использованием Python и Tkinter привлекали внимание ряда исследователей. Так, в статье Д. А. Вахидова, З. И. Махмудовой и Р. Ф. Сайфуллаева авторы рассматривают преимущества и недостатки Python и Tkinter для разработки приложений с GUI. Подробно анализируют области применения Tkinter в педагогической практике, акцентируя внимание на возможности создания кроссплатформенных решений, что делает их идеальными для использования в образовательных целях. В работе подчеркивается важность применения современных педагогических технологий с использованием этого инструмента [1].

Другим важным исследованием является работа авторов А. М. Вохидова, Д. А. Вохидова, Р. Ф. Фармоновой и Д. Ш. Хафизовой. В этой статье рассматривается использование Tkinter и Python в области медицины, где описаны ключевые аспекты создания медицинских приложений с графическим интерфейсом. Исследователи акцентируют внимание на удобстве использования Tkinter для создания интуитивно понятных интерфейсов для медицинских работников и пациентов. В работе подробно представлены примеры практических приложений, использующих эту технологию для создания удобных и доступных инструментов [2].

В исследовании К. Ювараджа, Г. М. Ураппана, К. К. Мегавартини рассмотрен вопрос использования Tkinter в сочетании с SQLite для разработки приложений для управления запасами. Авторы предлагают решение для перехода от традиционных методов учёта товаров к интегрированным цифровым системам, что значительно повышает эффективность работы. Приложение, описанное в исследовании, включает функции добавления и удаления сотрудников, обновления данных о запасах и создания резервных копий, что позволяет значительно снизить влияние человеческого фактора и повысить безопасность хранения данных [3].

В своей работе, авторы А. Нарендран, А. Хотри, М. Яшасвини исследуют возможности использования Tkinter и SQLite для создания персонализированных приложений для планирования путешествий. В статье

подробно рассмотрены особенности создания приложения для выбора маршрутов и отелей, а также другие функции, такие как сохранение планов поездок. Авторы акцентируют внимание на удобстве использования библиотеки Tkinter для реализации графических интерфейсов, а также на гибкости, которую предоставляет использование SQLite для хранения данных о пользователях и их маршрутах [4].

В своей статье А. С. Матвеева рассматривает применение паттерна «Команда» в создании модульных и структурированных приложений с использованием Tkinter. Автор демонстрирует, как этот паттерн позволяет создавать более гибкие и расширяемые приложения, где каждая кнопка выполняет отдельную команду. В результате такая реализация значительно облегчает процесс разработки и последующей модификации программ, повышая их надежность и удобство использования [5].

1.3 Цель исследования

Целью данного исследования является разработка и внедрение простого приложения "Todolist" с использованием Python и SQLite. Приложение должно поддерживать основные функции управления задачами, включая создание, редактирование, удаление задач, а также возможность смены аккаунта и регистрацию пользователей.

2 Материалы и методы

Для разработки приложения использовались следующие инструменты и технологии:

- Язык программирования Python версии 3.x.
- Библиотека Tkinter для создания графического интерфейса пользователя.
- SQLite для реализации базы данных, обеспечивающей хранение информации о пользователях и задачах.
- Модули Python для работы с базой данных (sqlite3) и управления пользовательским интерфейсом.

Основной метод разработки включал проектирование базы данных, создание логики взаимодействия с пользователем через графический интерфейс, а также интеграцию функционала регистрации и авторизации пользователей. В процессе разработки использовались принципы объектно-ориентированного программирования для улучшения структуры кода.

3 Результаты и обсуждения

Первым шагом в разработке приложения было проектирование базы данных для хранения информации о пользователях и задачах. Для этой цели была выбрана SQLite, так как она предоставляет легковесное решение для хранения данных и интегрируется с Python через стандартную библиотеку sqlite3 (рис.1).

Была создана база данных с двумя основными таблицами:

- users — таблица для хранения данных о пользователях (логин и пароль).
- tasks — таблица для хранения задач, связанных с пользователями. Каждая задача имеет текстовое описание и статус (например, "Ожидает", "Выполнено").

```
import sqlite3

def create_table(): 2 usages
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()

    # Создаем таблицу пользователей, если она не существует
    cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
)
''')

    # Создаем таблицу задач, если она не существует
    cursor.execute('''
CREATE TABLE IF NOT EXISTS tasks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    task TEXT NOT NULL,
    status TEXT NOT NULL,
    user_id INTEGER,
    FOREIGN KEY (user_id) REFERENCES users(id)
)
''')

    conn.commit()
    conn.close()
```

Рисунок 1. Код создания таблиц

После создания базы данных, следующим шагом стало добавление функционала для регистрации и авторизации пользователей. Это позволило обеспечить доступ к задачам только зарегистрированным пользователям, повысив безопасность и персонализацию работы с приложением (рис.2).

Для реализации функционала авторизации и регистрации была создана форма, на которой пользователи вводят свой логин и пароль. В базе данных осуществляется проверка на наличие пользователя. Если пользователь существует, он может войти в систему. Если пользователя нет, то предлагается зарегистрироваться.

```
def add_user(username, password): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'INSERT INTO users (username, password) VALUES (?, ?)', parameters: (username, password))
    conn.commit()
    conn.close()

def check_user(username, password): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'SELECT * FROM users WHERE username = ? AND password = ?', parameters: (username, password))
    user = cursor.fetchone()
    conn.close()
    return user
```

Рисунок 2. Код регистрации и авторизации

Далее был добавлен функционал добавления, удаления, получения и обновления задач каждого пользователя (рис.3).

```
def add_task(task, user_id): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'INSERT INTO tasks (task, status, user_id) VALUES (?, ?, ?)', parameters: (task, 'В работе', user_id))
    conn.commit()
    conn.close()

def get_tasks(user_id): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'SELECT * FROM tasks WHERE user_id = ?', parameters: (user_id,))
    tasks = cursor.fetchall()
    conn.close()
    return tasks

def delete_task(task_id): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'DELETE FROM tasks WHERE id = ?', parameters: (task_id,))
    conn.commit()
    conn.close()

def update_task_status(task_id, status): 1 usage
    conn = sqlite3.connect('tasks.db')
    cursor = conn.cursor()
    cursor.execute(sql: 'UPDATE tasks SET status = ? WHERE id = ?', parameters: (status, task_id))
    conn.commit()
    conn.close()
```

Рисунок 3. Код функционала для работы с задачами

Теперь перейдем к файлу main.py, который отвечает за реализацию графического интерфейса и взаимодействие с пользователем (рис.4-12).

```
import tkinter as tk
from tkinter import messagebox
import db
```

Рисунок 4. Импорт библиотек

В этом файле используются:

- tkinter — для создания графического интерфейса пользователя.
- messagebox — для отображения всплывающих окон с сообщениями.
- db — для работы с базой данных (импортируем файл db.py).

```
class TodoApp:
    def __init__(self, root, user_id):
        self.root = root
        self.user_id = user_id
        self.root.title('TodoList')
        self.root.geometry('400x500')

        # Обновляем таблицу задач
        db.create_table()

        # Виджеты
        self.task_entry = tk.Entry(self.root, width=30)
        self.task_entry.pack(pady=10)

        self.add_button = tk.Button(self.root, text='Добавить задачу', width=15, command=self.add_task)
        self.add_button.pack(pady=10)

        self.tasks_listbox = tk.Listbox(self.root, width=50, height=10)
        self.tasks_listbox.pack(pady=10)

        self.delete_button = tk.Button(self.root, text='Удалить задачу', width=15, command=self.delete_task)
        self.delete_button.pack(pady=5)

        self.update_button = tk.Button(self.root, text='Изменить статус', width=15, command=self.update_status)
        self.update_button.pack(pady=5)

        self.change_account_button = tk.Button(self.root, text='Сменить аккаунт', width=15, command=self.change_account)
        self.change_account_button.pack(pady=5)

        self.refresh_task_list()
```

Рисунок 5. Класс TodoApp

Конструктор класса:

- Инициализирует главный экран приложения, устанавливает заголовок окна и его размер.
- Создает виджеты для ввода задач, отображения списка задач, кнопки для добавления, удаления и изменения статуса задач.
- Кнопка "Сменить аккаунт" позволяет пользователю выйти из текущей сессии.

Методы:

- add_task — добавление новой задачи в базу данных.
- refresh_task_list — обновление списка задач на экране.
- delete_task — удаление задачи.
- update_status — изменение статуса задачи.
- change_account — возможность смены аккаунта.

```
def add_task(self): 1 usage
    task_text = self.task_entry.get()
    if task_text:
        db.add_task(task_text, self.user_id)
        self.task_entry.delete(first=0, tk.END)
        self.refresh_task_list()
    else:
        messagebox.showwarning(title='Ошибка ввода', message='Пожалуйста, введите задачу.')
```

Рисунок 6. Класс add_task

Этот метод добавляет новую задачу, используя функцию add_task из db.py. После добавления задачи поле ввода очищается, и список задач обновляется.

```
def refresh_task_list(self): 4 usages
    self.tasks_listbox.delete(first=0, tk.END)
    tasks = db.get_tasks(self.user_id)
    for task in tasks:
        task_text = f'{task[0]}: {task[1]} - {task[2]}'
        self.tasks_listbox.insert(tk.END, *elements: task_text)
```

Рисунок 7. Класс refresh_task

Этот метод обновляет список задач в пользовательском интерфейсе. Для этого он получает задачи из базы данных с помощью функции get_tasks из db.py и отображает их в Listbox.

```
def delete_task(self): 1 usage
    selected_task = self.tasks_listbox.curselection()
    if selected_task:
        task_id = int(self.tasks_listbox.get(selected_task[0]).split(':')[0])
        db.delete_task(task_id)
        self.refresh_task_list()
    else:
        messagebox.showwarning(title='Ошибка выбора', message='Пожалуйста, выберите задачу для удаления.')
```

Рисунок 8. Класс delete_task

Метод delete_task удаляет задачу из базы данных, если она была выбрана в списке. Алгоритм работы метода:

1. Получаем индекс выбранной задачи в Listbox с помощью метода curselection().
2. Извлекаем task_id из строки задачи, разделяя её по двоеточию, так как ID задачи находится в начале строки.
3. Вызываем функцию db.delete_task, передавая в неё task_id для удаления задачи из базы данных.
4. Обновляем список задач в интерфейсе через метод refresh_task_list.

Если задача не была выбрана, отображается предупреждающее сообщение.

```
def update_status(self): 1 usage
    selected_task = self.tasks_listbox.curselection()
    if selected_task:
        task_id = int(self.tasks_listbox.get(selected_task[0]).split(':')[0])
        new_status = 'Выполнено' if 'Ожидает' in self.tasks_listbox.get(selected_task[0]) else 'Ожидает'
        db.update_task_status(task_id, new_status)
        self.refresh_task_list()
    else:
        messagebox.showwarning( title: 'Ошибка выбора', message: 'Пожалуйста, выберите задачу для обновления статуса.')
```

Рисунок 9. Класс update_status

Метод update_status изменяет статус задачи с "Ожидает" на "Выполнено" и наоборот:

1. Получаем индекс выбранной задачи.
2. Извлекаем из строки текущий статус задачи (после символа "-").
3. Меняем статус задачи. Если статус был "Ожидает", то меняем на "Выполнено", иначе — на "Ожидает".
4. Обновляем статус задачи в базе данных с помощью db.update_task_status.
5. Обновляем список задач в интерфейсе.

Если задача не была выбрана, выводится предупреждающее сообщение.

```
def change_account(self): 1 usage
    self.root.destroy()
    root = tk.Tk()
    app = AuthApp(root)
    root.mainloop()
```

Рисунок 10. Класс change_account

Метод change_account позволяет пользователю выйти из текущего аккаунта и вернуться на экран авторизации:

1. Закрываем текущее окно с помощью self.root.destroy().
2. Создаем новое окно и запускаем приложение авторизации (AuthApp), которое будет отображать экран для входа или регистрации.

```

class AuthApp: 2 usages
    def __init__(self, root):
        self.root = root
        self.root.title('Авторизация')
        self.root.geometry('400x300')

        # Виджеты для авторизации
        self.username_label = tk.Label(self.root, text='Имя пользователя:')
        self.username_label.pack(pady=5)
        self.username_entry = tk.Entry(self.root, width=30)
        self.username_entry.pack(pady=5)

        self.password_label = tk.Label(self.root, text='Пароль:')
        self.password_label.pack(pady=5)
        self.password_entry = tk.Entry(self.root, width=30, show="*")
        self.password_entry.pack(pady=5)

        self.login_button = tk.Button(self.root, text='Войти', width=15, command=self.login)
        self.login_button.pack(pady=10)

        self.register_button = tk.Button(self.root, text='Зарегистрироваться', width=15, command=self.register)
        self.register_button.pack(pady=5)

    def login(self): 1 usage
        username = self.username_entry.get()
        password = self.password_entry.get()

        user = db.check_user(username, password)
        if user:
            self.root.destroy()
            root = tk.Tk()
            app = TodoApp(root, user[0]) # Передаем user_id
            root.mainloop()
        else:
            messagebox.showerror( title: 'Ошибка', message: 'Неверный логин или пароль.')

    def register(self): 1 usage
        username = self.username_entry.get()
        password = self.password_entry.get()

        if username and password:
            db.add_user(username, password)
            messagebox.showinfo( title: 'Успех', message: 'Регистрация прошла успешно. Вы можете войти.')
        else:
            messagebox.showwarning( title: 'Ошибка', message: 'Пожалуйста, заполните все поля.')

```

Рисунок 11. Класс AuthApp

Конструктор:

- Создает графический интерфейс для экрана авторизации с полями для ввода логина и пароля.
- Кнопки для входа и регистрации.

Методы:

- login: Проверяет введенные логин и пароль. Если они верны, то открывается окно задач.
- register: Создает нового пользователя в базе данных, если поля не пустые.

```
if __name__ == '__main__':  
    db.create_table()  
  
    root = tk.Tk()  
    app = AuthApp(root)  
    root.mainloop()
```

Рисунок 12. Код запуска приложения

Этот код запускает приложение, и его логика заключается в следующем:

1. Создание или проверка таблиц в базе данных.
2. Инициализация графического интерфейса.
3. Ожидание взаимодействия с пользователем (например, авторизация или регистрация).

Когда пользователь выполнит вход (или регистрацию), приложение продолжит работу с отображением задач, если вход прошел успешно.

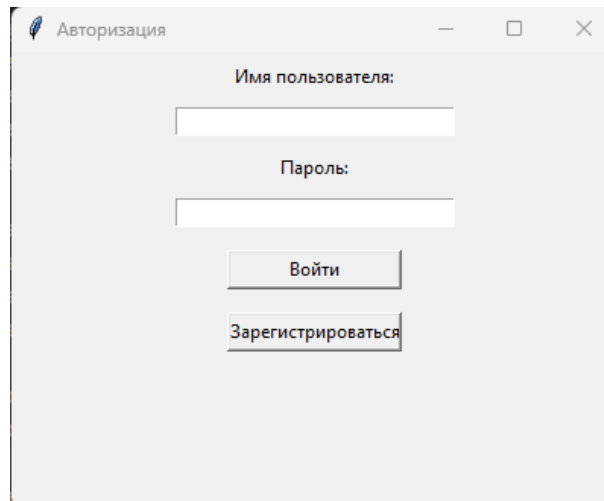


Рисунок 13. Внешний вид страницы авторизации и регистрации

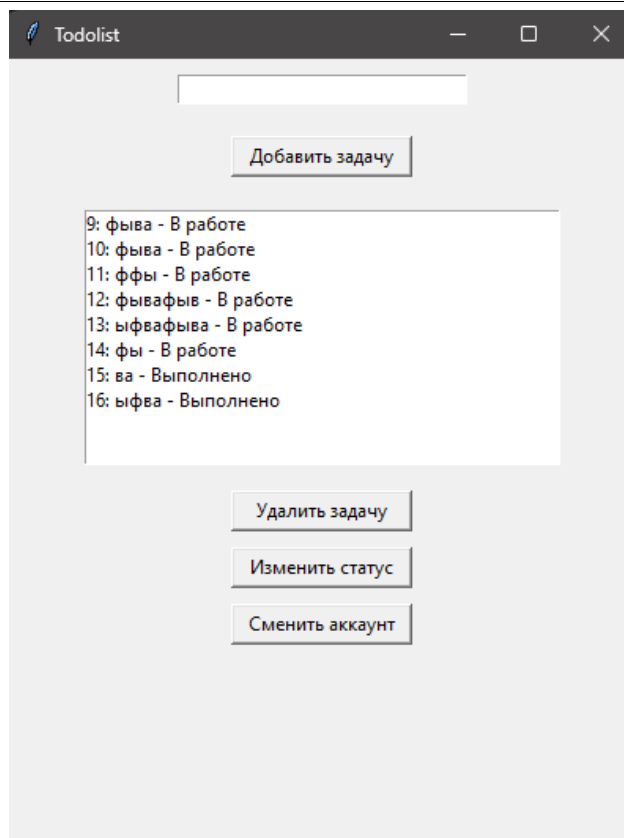


Рисунок 14. Внешний вид основной страницы приложения

4 Выводы

В ходе работы был разработан и реализован функционал для создания и управления приложением "Todolist" на языке Python с использованием базы данных SQLite для хранения задач пользователей. Приложение предоставляет пользователю возможность добавлять, редактировать, удалять и отслеживать задачи, а также поддерживает авторизацию и регистрацию через графический интерфейс на базе tkinter. В проекте был реализован механизм работы с базой данных для хранения пользователей и их задач, а также возможность смены аккаунта.

Особое внимание было уделено следующим аспектам:

Авторизация и регистрация пользователей: Встроенная система авторизации и регистрации обеспечивает высокий уровень безопасности, гарантируя, что только зарегистрированные пользователи могут взаимодействовать с задачами в приложении.

Интерфейс на языке Python с использованием tkinter: Разработанный графический интерфейс позволяет пользователю удобно управлять задачами, добавлять новые задачи, изменять их статус и удалять выполненные. Использование tkinter обеспечило простоту и эффективность при создании интерфейса.

Использование SQLite для хранения данных: Приложение эффективно использует SQLite для хранения информации о пользователях и задачах, что обеспечивает простоту в настройке и управлении базой данных, а также удобство работы с ней.

Гибкость и расширяемость: Приложение предоставляет возможность расширения функционала, например, добавления новых функциональных возможностей (например, категории задач или уведомлений), а также улучшений в безопасности, таких как хеширование паролей.

Решение, предложенное в рамках данной работы, является эффективным инструментом для ведения списка задач с возможностью пользовательской авторизации и регистрации, а также хранения данных в удобной базе данных SQLite. Система полностью адаптирована для пользователей, и её функционал отвечает стандартам современных приложений для управления задачами.

В дальнейшем возможна оптимизация и добавление новых функций, таких как поддержка нескольких пользователей в одной сессии, улучшение интерфейса и обеспечение более строгой безопасности данных, включая хеширование паролей и внедрение двухфакторной аутентификации.

Библиографический список

1. Aligulovich D.A., Mahmudova Z.I., Sayfullayev R.F. Tibbiyot yo'nalishida zamonaviy pedagogik texnologiyalarini qo'llab tkinter paketida gui dasturlarini tuzish // Евразийский журнал математической теории и компьютерных наук. 2022. №2(12). С. 31-35.
2. Вохидов А.М., Вохидов Д.А., Фармонова Р.Ф., Хафизова Д.Ш. Разработка графическим пользовательским интерфейсом -программ в пакете tkinter с использованием современных педагогических технологий в области медицины // Роль информационных технологий в управлении ресурсами персонала здравоохранения. 2022. Т. 34.
3. Yuvaraj K., Oorappan G.M., Megavarthini K.K. Design And Development Of An Application For Database Maintenance In Inventory Management System Using Tkinter And Sqlite Platform // IOP Conference Series: Materials Science and Engineering. 2020. Т. 995.
4. Narendran A., Hothri A., Yashaswini M. Personalized Travel Itinerary Recommender System using Tkinter and SQLite // 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). 2024.
5. Матвеева А.С Реализация паттерна поведения «Команда» в графическом интерфейсе пользователя на Python // Постулат. 2025. №1.