

Создание приложения «Заметки» на фреймворке Kivy

Агентьев Данила Денисович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В работе представлено приложение "Заметки", разработанное с использованием фреймворка Kivy. Приложение включает функционал для регистрации, аутентификации пользователей, а также для управления заметками. Основной целью разработки было создание кроссплатформенного приложения для хранения и редактирования заметок с возможностью взаимодействия с базой данных через SQLite. В статье рассматривается структура проекта, основные этапы разработки, а также особенности работы с графическим интерфейсом пользователя и базой данных.

Ключевые слова: Kivy, приложение для заметок, регистрация, аутентификация, база данных, SQLite, пользовательский интерфейс, мобильные приложения, Python.

Creating the «Notes» application on the Kivy framework

Akentev Danila Denisovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This paper presents the Notes application, developed using the Kivy framework. The application includes functionality for registration, user authentication, and note management. The main goal of the development was to create a cross-platform application for storing and editing notes with the ability to interact with a database via SQLite. The article discusses the structure of the project, the main stages of development, as well as the features of working with a graphical user interface and a database.

Keywords: Kivy, note-taking application, registration, authentication, database, SQLite, user interface, mobile applications, Python.

1 Введение

1.1 Актуальность

Современные мобильные и десктопные приложения требуют удобных и интуитивно понятных интерфейсов, а также надежной работы с данными. Приложение "Заметки" решает задачу хранения и управления текстовой информацией в форме заметок, а также предоставляет функционал для регистрации и аутентификации пользователей. Использование фреймворка

Kivy для разработки кроссплатформенных приложений актуально, поскольку оно позволяет создавать приложения для различных операционных систем, включая Android и iOS.

1.2 Обзор исследований

В статье И. Ф. Андерсона представлено приложение QRGB++, использующее метод аддитивного цвета RGB для создания расширенных QR-кодов, что позволяет хранить в три раза больше информации благодаря объединению трёх цветовых слоёв (красного, зелёного и синего). Ключевым моментом работы является использование фреймворка Kivy для создания удобного графического интерфейса, который облегчает взаимодействие пользователя с приложением. В качестве дополнительных технологий для обработки изображений используются библиотеки Pillow, opencv-python и qrcode[pil], что позволяет эффективно создавать и декодировать QR-коды. Несмотря на проблемы с точностью извлечения данных и смешиванием цветов, подход значительно повышает плотность хранимой информации и имеет потенциал для применения в таких сферах, как безопасность, реклама и визуальная криптография [1].

В работе А. Ф. М. Гада рассматривается использование языка KV для разделения логики и графического интерфейса в приложениях на Kivy. В рамках работы автор объясняет процесс подготовки среды разработки для создания кроссплатформенных приложений, включая установку необходимых инструментов, таких как Buildozer для сборки мобильных приложений. Основное внимание уделяется тому, как с помощью Kivy можно использовать один и тот же код Python для создания приложений как для настольных, так и для мобильных устройств, благодаря поддержке кроссплатформенности библиотеки. Автор также описывает подготовку Android SDK, NDK и необходимых библиотек Python для успешной сборки приложения для Android, что подтверждает гибкость и мощь Kivy как инструмента для создания мобильных и настольных приложений с разделением логики и интерфейса [2].

К. Катпала, Ш. Неги и Ш. Шармы представили в своей работе инновационную систему iChat, использующую технологию распознавания взгляда для людей с особыми потребностями. С помощью библиотеки OpenCV и фреймворка Kivy на Python реализовано приложение, которое позволяет пользователю взаимодействовать с компьютером с помощью глаз, что особенно важно для людей с ограниченными физическими возможностями. В основе системы лежит использование 68 точек на лице для точного определения положения глаз, что повышает точность распознавания. Разделение виртуальной клавиатуры на левую и правую части упрощает процесс ввода текста с помощью моргания, а добавление звуковых эффектов с использованием модуля Pyglet делает приложение более интуитивно понятным. Это решение является значительным шагом в направлении создания технологий для улучшения качества жизни людей с ограниченными

возможностями и предоставляет новую возможность для общения и набора текста без использования рук [3].

В работе И. С. Йи представлен проект "DealWithIt", который использует компьютерное зрение, машинное обучение и мобильное приложение для решения проблемы сравнения цен и получения информации о товарах в реальном времени. Проект включает в себя систему распознавания объектов, которая позволяет автоматически идентифицировать товары и измерять их размеры с помощью технологий компьютерного зрения (CV). Используя веб-скрапинг для извлечения актуальной информации о ценах с онлайн-магазинов, проект поддерживает базу данных, которая регулярно обновляется, обеспечивая точность и актуальность данных. Модель машинного обучения распознает тип продукта и извлекает информацию из базы данных, позволяя пользователю легко и быстро получить необходимые сведения о товаре через мобильное приложение. Все эти функции — обнаружение объектов, распознавание товаров, измерение размеров и обновление цен — интегрируются в удобный интерфейс на платформе Kivy, обеспечивая пользователям простой и эффективный способ контроля за покупками в супермаркетах [4].

Кроме того, в последние годы Python активно используется для разработки мобильных и настольных приложений. Д.В. Лучанинов и А.О. Кизянов исследовали создание игры 2048 на фреймворке Kivy, что продемонстрировало практическую значимость Python для разработки графических интерфейсов и приложений с использованием современных инструментов [5].

1.3 Цель исследования

Целью исследования является разработка и описание приложения для управления заметками с использованием фреймворка Kivy, а также внедрение базовых функций взаимодействия с базой данных (SQLite) для хранения и редактирования заметок, а также обеспечения безопасности с помощью аутентификации и регистрации пользователей.

2 Материалы и методы

Для разработки приложения использовались следующие материалы и методы:

- Фреймворк *Kivy* — для создания графического интерфейса пользователя.
- *KivyMD* — для улучшения пользовательского интерфейса с использованием Material Design.
- *SQLite* — для работы с базой данных, хранения пользователей и заметок.
- *bcrypt* — для хеширования паролей пользователей.
- *Python* — основной язык программирования.

Разработка велась с использованием паттерна проектирования MVC (Model-View-Controller), где:

- *Model* представляет собой базу данных.
- *View* — пользовательский интерфейс.
- *Controller* — обработчики событий, такие как регистрация, аутентификация, добавление, редактирование и удаление заметок.

3 Результаты и обсуждения

Структура проекта выглядит следующим образом

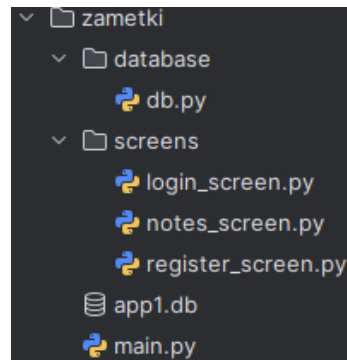


Рисунок 1. Структура проекта

Файл `main.py`: этот файл запускает приложение, инициализирует базу данных и настраивает экранный менеджер для переключения между экранами (вход, регистрация, заметки). Он импортирует необходимые экраны и базовые функции для создания таблиц в базе данных (пользователи и заметки).

```
from kivymd.app import MDApp
from kivy.uix.screenmanager import ScreenManager
from screens.login_screen import LoginScreen
from screens.register_screen import RegisterScreen
from screens.notes_screen import NotesScreen
from database.db import create_user_table, create_notes_table

class NotesApp(MDApp): 1 usage
    def build(self):
        # Создание базы данных (если она еще не существует)
        create_user_table()
        create_notes_table()

        # Менеджер экранов
        sm = ScreenManager()
        sm.add_widget(LoginScreen(name="login"))
        sm.add_widget(RegisterScreen(name="register"))
        sm.add_widget(NotesScreen(name="notes"))

        return sm

if __name__ == "__main__":
    NotesApp().run()
```

Рисунок 2. Код `main.py`

Данный файл является точкой входа в приложение. Основная цель — это создать и запустить приложение с экранным менеджером для переключения между экранами:

- В методе *build()* создаются необходимые таблицы базы данных (если они еще не существуют), используя функции *create_user_table()* и *create_notes_table()*.
- Затем создается объект *ScreenManager*, который будет управлять различными экранами (вход, регистрация, заметки).
- Каждый экран добавляется в менеджер экранов с уникальным именем.
- Приложение запускается с помощью *NotesApp().run()*, что активирует работу всех экранов и компонентов.

Файл *register_screen.py*: экран регистрации, на котором пользователи могут ввести логин и пароль. При успешной регистрации данные сохраняются в базе данных. Также реализована кнопка для перехода назад к экрану входа.

```
from kivymd.uix.screen import Screen
from kivymd.uix.textfield import MDTextField
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.dialog import MDDialog
from database.db import register_user

class RegisterScreen(Screen): 2 usages
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.name = "register"

        self.username_field = MDTextField(
            hint_text="Введите логин",
            size_hint=(0.8, None),
            height=50,
            pos_hint={"center_x": 0.5, "center_y": 0.6}
        )
        self.add_widget(self.username_field)

        self.password_field = MDTextField(
            hint_text="Введите пароль",
            size_hint=(0.8, None),
            height=50,
            pos_hint={"center_x": 0.5, "center_y": 0.5},
            password=True
        )
        self.add_widget(self.password_field)

        self.register_button = MDRaisedButton(
            text="Зарегистрироваться",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.4},
            on_release=self.register
        )
        self.add_widget(self.register_button)

        self.back_button = MDRaisedButton(
            text="Назад",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.3},
            on_release=self.go_back_to_login
        )
        self.add_widget(self.back_button)
```

Рисунок 3.1. Код *register_screen.py*

```
def register(self, instance): 1 usage
    username = self.username_field.text
    password = self.password_field.text
    if username and password:
        register_user(username, password)
        self.manager.current = "login"
    else:
        self.show_error_dialog("Пожалуйста, заполните все поля.")

def go_back_to_login(self, instance): 1 usage
    self.manager.current = "login"

def show_error_dialog(self, message): 1 usage
    dialog = MDDialog(
        title="Ошибка",
        text=message,
        buttons=[MDRaisedButton(text="Закрыть", on_release=lambda x: dialog.dismiss())]
    )
    dialog.open()
```

Рисунок 3.2. Код register_screen.py

Данный файл описывает экран регистрации. На экране два поля ввода: для логина и пароля. Также есть две кнопки:

- Кнопка для регистрации вызывает метод *register()*, который проверяет, что оба поля заполнены, и сохраняет данные в базе данных с помощью функции *register_user()*.
- Кнопка "Назад" позволяет вернуться на экран входа, переключая экран с помощью менеджера экранов (*self.manager.current = "login"*).
- В случае ошибки (например, пустые поля) появляется диалог с ошибкой.

Файл *notes_screen.py*: экран для отображения и управления заметками. Здесь пользователи могут добавлять, редактировать, удалять и просматривать свои заметки. Все действия сохраняются в базе данных. При редактировании заметки открывается диалоговое окно с возможностью сохранить или удалить изменения.

```

from kivymd.uix.screen import Screen
from kivymd.uix.textfield import MDTextField
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.list import MDList, OneLineListItem
from kivy.uix.scrollview import ScrollView
from kivymd.uix.dialog import MDDialog
from functools import partial
from database.db import load_notes_from_db, save_note_to_db, delete_note_from_db, update_note_in_db

class NotesScreen(Screen): 2 usages
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.name = "notes"
        self.user_id = None

        # Поле ввода заметки
        self.text_field = MDTextField(
            hint_text="Введите заметку",
            size_hint=(0.8, None),
            height=50,
            pos_hint={"center_x": 0.5, "center_y": 0.9}
        )
        self.add_widget(self.text_field)

        # Кнопка добавления заметки
        self.add_button = MDRaisedButton(
            text="Добавить заметку",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.8},
            on_release=self.add_note,
        )
        self.add_widget(self.add_button)

        # Прокручиваемый список заметок
        self.scroll_view = ScrollView(pos_hint={"center_x": 0.5, "center_y": 0.4}, size_hint=(0.9, 0.6))
        self.notes_list = MDList()
        self.scroll_view.add_widget(self.notes_list)
        self.add_widget(self.scroll_view)

        # Кнопка выхода
        self.logout_button = MDRaisedButton(
            text="Выйти",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.1},
            on_release=self.logout,
        )
        self.add_widget(self.logout_button)

    def set_user_id(self, user_id): 1 usage (1 dynamic)
        """Метод для установки ID пользователя и загрузки его заметок"""
        self.user_id = user_id
        self.notes = load_notes_from_db(user_id)
        self.update_notes_list()

```

Рисунок 4.1. Код notes_screen.py

```

def add_note(self, instance): 1 usage
    """Метод для добавления новой заметки"""
    note_text = self.text_field.text.strip()
    if note_text:
        save_note_to_db(self.user_id, note_text)
        self.notes.append(note_text)
        self.update_notes_list()
        self.text_field.text = ""

def update_notes_list(self): 4 usages
    """Метод для обновления списка заметок"""
    self.notes_list.clear_widgets()
    for note in self.notes:
        item = OneLineListItem(
            text=note,
            on_release=partial(self.edit_note, note_text=note) # Используем partial
        )
        self.notes_list.add_widget(item)

def edit_note(self, instance, note_text): 1 usage
    """Метод для редактирования заметки"""
    # Создаем поле для редактирования заметки, обернутого в ScrollView для прокрутки
    content = ScrollView(
        size_hint=(1, None),
        height=200,
        do_scroll_x=False # Отключаем горизонтальную прокрутку
    )

    text_field = MDTextField(
        text=note_text,
        multiline=True,
        size_hint=(1, None),
        height=200
    )

    content.add_widget(text_field)

    # Добавляем кнопку "Удалить" в диалог
    delete_button = MDRaisedButton(
        text="Удалить",
        size_hint=(None, None),
        size=(100, 50),
        pos_hint={"right": 1},
        on_release=lambda x: self.delete_note(note_text, dialog) # Передаем диалог в метод
    )

    # Кнопка "Сохранить"
    save_button = MDRaisedButton(
        text="Сохранить",
        on_release=lambda x: self.save_edited_note(text_field.text, note_text, dialog)
    )

```

Рисунок 4.2. Код notes_screen.py

```

dialog = MDDialog(
    title="Редактировать заметку",
    type="custom",
    content_cls=content,
    buttons=[
        save_button,
        delete_button, # Добавляем кнопку удаления в диалог
        MDRaisedButton(
            text="Отменить",
            on_release=lambda x: dialog.dismiss()
        ),
    ],
)
dialog.open()

def save_edited_note(self, new_text, old_text, dialog): 1 usage
    """Метод для сохранения изменённой заметки"""
    if new_text != old_text and new_text.strip():
        update_note_in_db(self.user_id, old_text, new_text)
        self.notes = load_notes_from_db(self.user_id)
        self.update_notes_list()
        dialog.dismiss()

def delete_note(self, note_text, dialog): 1 usage
    """Метод для удаления заметки"""
    delete_note_from_db(self.user_id, note_text)
    self.notes = load_notes_from_db(self.user_id)
    self.update_notes_list()

    # Закрытие диалога после удаления
    dialog.dismiss()

def logout(self, instance): 1 usage
    """Метод для выхода из приложения"""
    self.manager.current = "login"

```

Рисунок 4.3. Код notes_screen.py

Данный файл описывает экран с заметками, где пользователи могут:

- Просматривать свои заметки, добавлять новые, редактировать и удалять существующие.
- При клике на заметку открывается диалоговое окно для редактирования или удаления заметки.
- Метод `set_user_id()` загружает заметки для конкретного пользователя при его аутентификации.
- Все изменения сохраняются в базе данных с помощью функций `save_note_to_db()`, `update_note_in_db()`, и `delete_note_from_db()`.
- Кнопка "Выйти" возвращает пользователя на экран входа.

Файл `login_screen.py`: экран входа, где пользователи вводят свой логин и пароль. При успешной аутентификации приложение перенаправляет пользователя на экран с заметками. В случае ошибки отображается диалог с уведомлением.

```
from kivymd.uix.screen import Screen
from kivymd.uix.textfield import MDTextField
from kivymd.uix.button import MDRaisedButton
from kivymd.uix.dialog import MDDialog
from database.db import authenticate_user, get_user_id

class LoginScreen(Screen): 2 usages
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.name = "login"

        self.username_field = MDTextField(
            hint_text="Введите логин",
            size_hint=(0.8, None),
            height=50,
            pos_hint={"center_x": 0.5, "center_y": 0.6}
        )
        self.add_widget(self.username_field)

        self.password_field = MDTextField(
            hint_text="Введите пароль",
            size_hint=(0.8, None),
            height=50,
            pos_hint={"center_x": 0.5, "center_y": 0.5},
            password=True
        )
        self.add_widget(self.password_field)

        self.login_button = MDRaisedButton(
            text="Войти",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.4},
            on_release=self.login
        )
        self.add_widget(self.login_button)

        self.register_button = MDRaisedButton(
            text="Регистрация",
            size_hint=(0.5, None),
            pos_hint={"center_x": 0.5, "center_y": 0.3},
            on_release=self.go_to_register
        )
        self.add_widget(self.register_button)
```

Рисунок 5.1. Код login_screen.py

```
def login(self, instance): 1 usage
    username = self.username_field.text
    password = self.password_field.text
    if authenticate_user(username, password):
        user_id = get_user_id(username)
        self.manager.current = "notes"
        self.manager.get_screen("notes").set_user_id(user_id)
    else:
        self.show_error_dialog("Неверные данные для входа.")

def go_to_register(self, instance): 1 usage
    self.manager.current = "register"

def show_error_dialog(self, message): 1 usage
    dialog = MDDialog(
        title="Ошибка",
        text=message,
        buttons=[MDRaisedButton(text="Закрыть", on_release=lambda x: dialog.dismiss())]
    )
    dialog.open()
```

Рисунок 5.2. Код login_screen.py

Данный файл описывает экран входа. На экране два поля ввода: для логина и пароля, а также две кнопки:

- Кнопка "Войти":
 - Вызывает метод `login()`, который проверяет правильность введенных данных (логина и пароля).
 - Если данные правильные, то происходит переход на экран заметок (экран с заметками).
 - В случае неверного логина/пароля появляется диалог с ошибкой.
- Кнопка "Зарегистрироваться":
 - Перекидывает пользователя на экран регистрации, переключая экран с помощью менеджера экранов (`self.manager.current = "register"`).
- Метод `show_error_dialog()`:
 - Показывает диалоговое окно с ошибкой в случае, если введены неправильные данные или поля пустые.

Файл `db.py`: в этом файле содержатся функции для взаимодействия с базой данных SQLite. Основные функции включают создание таблиц, регистрацию пользователей, аутентификацию, загрузку, сохранение, редактирование и удаление заметок. Для повышения безопасности пароли пользователей хешируются с использованием библиотеки `bcrypt`.

```
import bcrypt
import sqlite3

# Функции для работы с базой данных
def create_user_table(): 2 usages
    conn = sqlite3.connect('appl.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE,
            password TEXT
        )
    ''')
    conn.commit()
    conn.close()

def create_notes_table(): 2 usages
    conn = sqlite3.connect('appl.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS notes (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            note_text TEXT,
            FOREIGN KEY(user_id) REFERENCES users(id)
        )
    ''')
    conn.commit()
    conn.close()

def register_user(username, password):
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
    conn = sqlite3.connect('appl.db')
    cursor = conn.cursor()
    cursor.execute(sql: "INSERT INTO users (username, password) VALUES (?, ?)", parameters: (username, hashed_password))
    conn.commit()
    conn.close()

def authenticate_user(username, password):
    conn = sqlite3.connect('appl.db')
    cursor = conn.cursor()
    cursor.execute(sql: "SELECT password FROM users WHERE username=?", parameters: (username,))
    result = cursor.fetchone()
    conn.close()
    if result and bcrypt.checkpw(password.encode('utf-8'), result[0]):
        return True
    return False
```

Рисунок 6.1. Код db.py

```
def get_user_id(username):
    conn = sqlite3.connect('app1.db')
    cursor = conn.cursor()
    cursor.execute(sql: "SELECT id FROM users WHERE username=?", parameters: (username,))
    result = cursor.fetchone()
    conn.close()
    if result:
        return result[0]
    return None

def load_notes_from_db(user_id):
    conn = sqlite3.connect('app1.db')
    cursor = conn.cursor()
    cursor.execute(sql: "SELECT note_text FROM notes WHERE user_id=?", parameters: (user_id,))
    notes = [row[0] for row in cursor.fetchall()]
    conn.close()
    return notes

def save_note_to_db(user_id, note_text):
    conn = sqlite3.connect('app1.db')
    cursor = conn.cursor()
    cursor.execute(sql: "INSERT INTO notes (user_id, note_text) VALUES (?, ?)", parameters: (user_id, note_text))
    conn.commit()
    conn.close()

def delete_note_from_db(user_id, note_text):
    conn = sqlite3.connect('app1.db')
    cursor = conn.cursor()
    cursor.execute(sql: "DELETE FROM notes WHERE user_id=? AND note_text=?", parameters: (user_id, note_text))
    conn.commit()
    conn.close()

def update_note_in_db(user_id, old_text, new_text):
    conn = sqlite3.connect('app1.db')
    cursor = conn.cursor()
    cursor.execute(sql: "UPDATE notes SET note_text=? WHERE user_id=? AND note_text=?", parameters: (new_text, user_id, old_text))
    conn.commit()
    conn.close()

# Функция для обновления данных пользователя
def update_user_data(new_username):
    conn = sqlite3.connect('notes.db')
    cursor = conn.cursor()
    cursor.execute(sql: "UPDATE Users SET username = ? WHERE id = ?", parameters: (new_username, 1)) # 1 - это ID текущего пользователя
    conn.commit()
    conn.close()
```

Рисунок 6.2. Код db.py

Файл отвечает за взаимодействие с базой данных SQLite, содержащей две таблицы:

- *users* — для хранения данных пользователей.
- *notes* — для хранения заметок, связанных с пользователем.

Каждая функция выполняет конкретную операцию: создание таблиц, регистрация пользователей, сохранение, редактирование и удаление заметок.

- Функции для работы с паролями используют *bcrypt* для хеширования.
- Все изменения в базе данных подтверждаются вызовом *commit()* и выполняются с помощью SQL-запросов.

4 Выводы

В ходе работы были рассмотрены основные аспекты разработки приложения для создания и управления заметками с использованием фреймворка Kivy и языка программирования Python. Предложенная система включает в себя такие ключевые компоненты, как регистрация и аутентификация пользователей, хранение и управление заметками с возможностью их редактирования и удаления. Эти функции были реализованы с применением объектно-ориентированного подхода, что

позволяет расширять функциональные возможности приложения и облегчает его поддержку и масштабирование.

Особое внимание уделено интеграции Python с базой данных SQLite для хранения пользовательских данных и заметок, что продемонстрировало практическую эффективность данного языка в разработке приложений с графическим интерфейсом и взаимодействием с базами данных. Реализация всех операций с данными, таких как регистрация пользователей, сохранение, редактирование и удаление заметок, позволяет создать удобную и функциональную среду для пользователя.

Использование фреймворка Kivy в проекте также продемонстрировало его возможности для создания кросс-платформенных приложений с минимальными усилиями. В результате разработка приложения подтверждает высокую эффективность Python и Kivy как инструментов для создания приложений с пользовательскими интерфейсами, которые могут работать на различных операционных системах, включая Windows, macOS и Linux.

Таким образом, разработка приложения заметок на фреймворке Kivy является наглядным примером использования Python в реальных задачах программирования. Это подтверждает практическую значимость Python как универсального инструмента для создания простых и сложных приложений в самых различных областях, включая образование, инженерные дисциплины и прикладное программирование.

Библиографический список

1. Anderson I. F. QRGB++ in Python running in Visual Studio Code with a graphical interface (pip install kivy) + pip install pillow + pip install qrcode[pil] + pip install opencv-python // Facultad de Artes. 2024 C. 1-17
2. Gad A. F. M. Using KV Language for Separation of Logic and GUI // Building Android Apps in Python Using Kivy with Android Studio. 2019 C. 21-37
3. Kathpal K., Negi S., Sharma S. iChat: Interactive Eyes for Specially Challenged People Using OpenCV Python // 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). 2021
4. Yee Y. E. DealWithIt - Real Time Price Checker with Object Recognition // Universiti Tunku Abdul Rahman in partial fulfillment of the requirements for the degree of BACHELOR OF COMPUTER SCIENCE (HONOURS) Faculty of Information and Communication Technology (Kampar Campus). 2024.
5. Кизянов А.О., Лучанинов Д.В. Разработка игры 2048 на фреймворке Kivy // Постулат. 2016 № 9