

Разработка телеграмм-бота

Birthday reminder



выполнила ученица курса

«Основы промышленного
программирования»

Яндекс Лицея

Русина Лидия

СТРУКТУРА ПРОГРАММЫ

1. `main.py` - запуск программы, код бота
2. `database.py` - работа с базой данных
3. `config.py` - получение токена бота и url базы данных
4. `util.py` - дополнительные методы, например, создание клавиатуры

► Дополнительные файлы

1. `README.md` - справочная информация
2. `birthdays.db` - база данных
3. `requirements.txt` - необходимые библиотеки

► Используемые технологии

Библиотеки: `logging`, `logging`, `schedule`, `threading`, `asyncio`, `time`, `sqlalchemy`, `datetime`.

Телеграм-бот, напоминающий о днях рождениях

Данный бот хранит данные о днях рождениях, добавленных пользователем. Можно вывести список всех дней рождений, посмотреть у кого день рождение сегодня, добавлять и удалять записи. Также в тот день, когда у пользователя есть запись с днем рождения, он получит автоматическое напоминание о нем в 9 утра.

Управление

Найдите бота по никнейму `@cuteshka_bot`

Стартовое меню

`/help` или `/start`

Выбор просмотра или редактирования

с помощью кнопок, привязанных к сообщению

Добавление или удаление записи

нажатием в тексте или вводом сообщений `/add` `/delete`

Выбор нужной записи для удаления

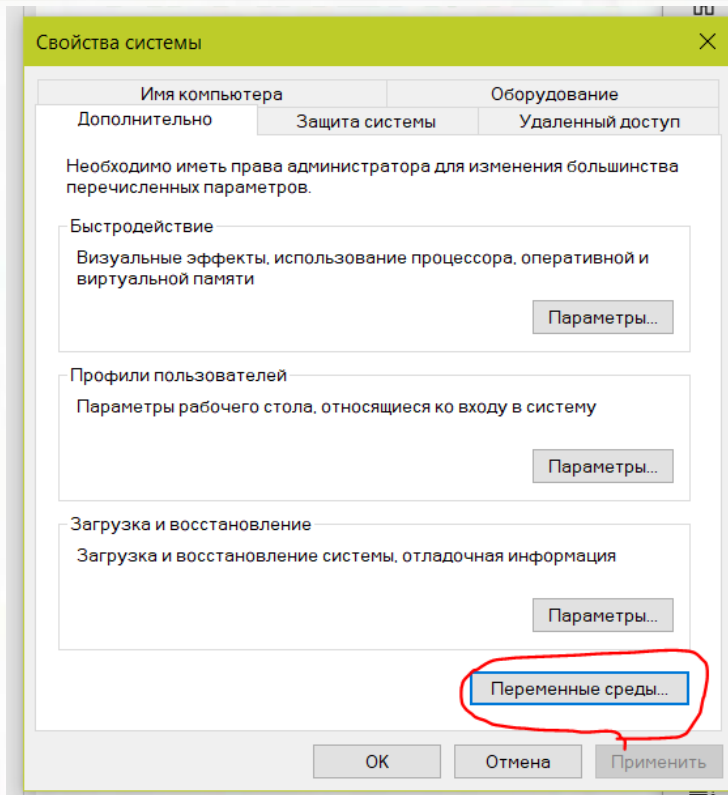
из списка во всплывающем меню

ХРАНЕНИЕ ТОКЕНА В ПЕРЕМЕННОЙ ОКРУЖЕНИЯ

```
import os
```

```
BOT_TOKEN = os.environ.get("BOT_TOKEN")
```

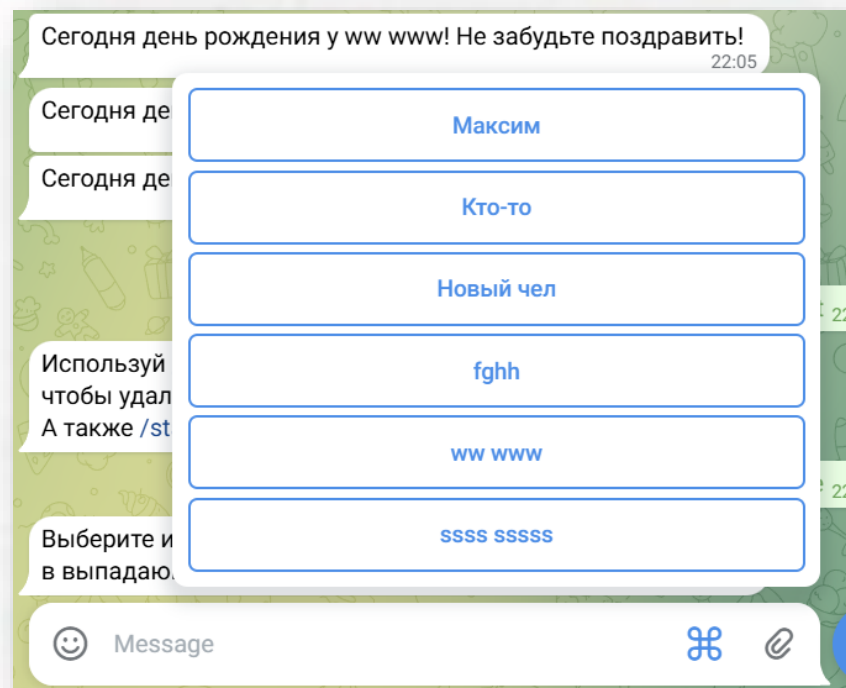
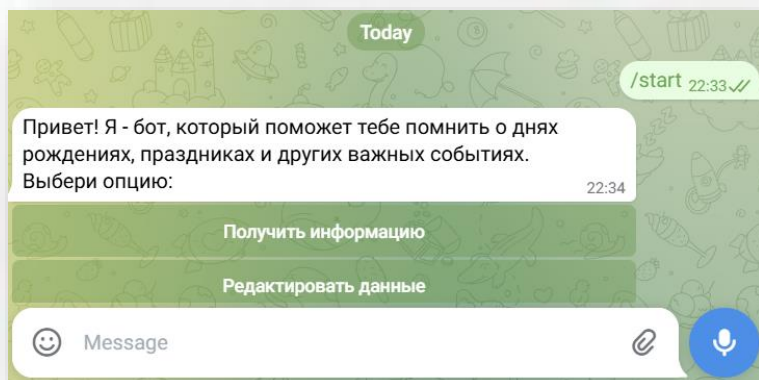
```
DATABASE_URL = os.environ.get("DATABASE_URL", "sqlite:///birthdays.db")
```



- ▶ Токен не хранится непосредственно в коде приложения. Это снижает риск случайной утечки токена в систему контроля версий (например, Git), где он может стать общедоступным.
- ▶ Переменные окружения - это стандартный механизм для конфигурации приложений во многих операционных системах и платформах развертывания.
- ▶ Использование переменных окружения для хранения чувствительной информации, такой как токены, является общепринятой лучшей практикой разработки безопасных приложений.

ОСОБЕННОСТИ ПРОГРАММЫ

- ▶ **Создание различных видов клавиатур:**
InlineKeyboardMarkup,
ReplyKeyboardMarkup
- ▶ **обработка диалога с пользователем с помощью ConversationHandler.**



```
# ConversationHandler для добавления дня рождения
add_birthday_handler = ConversationHandler(
    entry_points=[CommandHandler(command="add", add_birthday_start)],
    states={
        ADD_NAME: [MessageHandler(filters.TEXT & ~filters.COMMAND, add_birthday_surname_name)],
        ADD_GROUP: [MessageHandler(filters.TEXT & ~filters.COMMAND, add_group)],
        ADD_DETAILS: [MessageHandler(filters.TEXT & ~filters.COMMAND, add_details)],
        ADD_DATE: [MessageHandler(filters.TEXT & ~filters.COMMAND, add_birthday_date)],
    },
    fallbacks=[CommandHandler(command="cancel", add_birthday_cancel)],
)
application.add_handler(add_birthday_handler)
```

ОСОБЕННОСТИ ПРОГРАММЫ

- Создание и работа с базой данных с помощью sqlalchemy (создание, редактирование, фильтрация данных)






```
engine = create_engine(DATABASE_URL, echo=False)
SqlAlchemyBase = orm.declarative_base()
__factory = None
```

```
class Birthday(SqlAlchemyBase):
    __tablename__ = 'birthdays'
```

```
    id = Column(Integer, primary_key=True, autoincrement=True)
    user_id = Column(Integer, nullable=False)
    surname_name = Column(String, nullable=False)
    date = Column(Date, nullable=False)
    group = Column(String, nullable=True)
    details = Column(String, nullable=True)
```

```
    def __repr__(self):
        return f"<Birthday(surname_name='{self.surname_name}', date='{self.date}')>"
```

```
if not __factory:
    __factory = orm.sessionmaker(bind=engine)
    SqlAlchemyBase.metadata.create_all(engine)
```

birthdays		Имя таблицы:	birthdays					<input type="checkbox"/> WITHOUT ROWID	<input type="checkbox"/> STRICT
	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL		
1	id	INTEGER							
2	user_id	INTEGER							
3	surname_name	VARCHAR							
4	date	DATE							
5	group	VARCHAR							
6	details	VARCHAR							

```
def run_async_job():
    asyncio.run(scheduled_job())
```

```
schedule.every().day.at("09:00").do(run_async_job)
```

```
def scheduler_loop():
    while True:
        schedule.run_pending()
        time.sleep(1)
```

```
scheduler_thread = threading.Thread(target=scheduler_loop, daemon=True)
scheduler_thread.start()
```

- Использование библиотеки **schedule** и **threading** для создания автоматических уведомлений в отдельном потоке

ВОЗМОЖНОСТИ ДЛЯ ДОРАБОТКИ

- ▶ Добавление оповещения о праздниках с предзагрузкой основных, возможностью догрузить остальные автоматически (например, религиозные), добавлять и удалять отдельные записи;
- ▶ Добавление отдельного списка важных дел и событий с возможностью настройки времени и количества оповещений;
- ▶ Автоматическая отправка поздравлений с днями рождения и праздниками пользователям с возможностью использовать предзагруженные шаблоны для отдельных групп (родственники, коллеги, друзья);
- ▶ Генерация открыток с пользовательским или предзагруженным текстом.

СООТВЕТСТВИЕ КРИТЕРИЯМ

- ▶ Использован основной функционал, изученный на курсе
 - ▶ Использован дополнительный функционал, изученный самостоятельно (многопоточность; создание меню, привязанного к сообщению, и обработка нажатия кнопок; хранение токена в переменной окружения)
 - ▶ Код разделен на классы, файлы удобно сгруппированы.
 - ▶ Код соответствует PEP 8, константы вынесены в отдельный файл, использованы «говорящие» имена переменных.
 - ▶ Создан файл requirements.txt, README.md с пояснениями к программе;
 - ▶ Проект сохранен на github
- пошагово с пояснениями;
- ▶ более 400 строк функционального кода.

