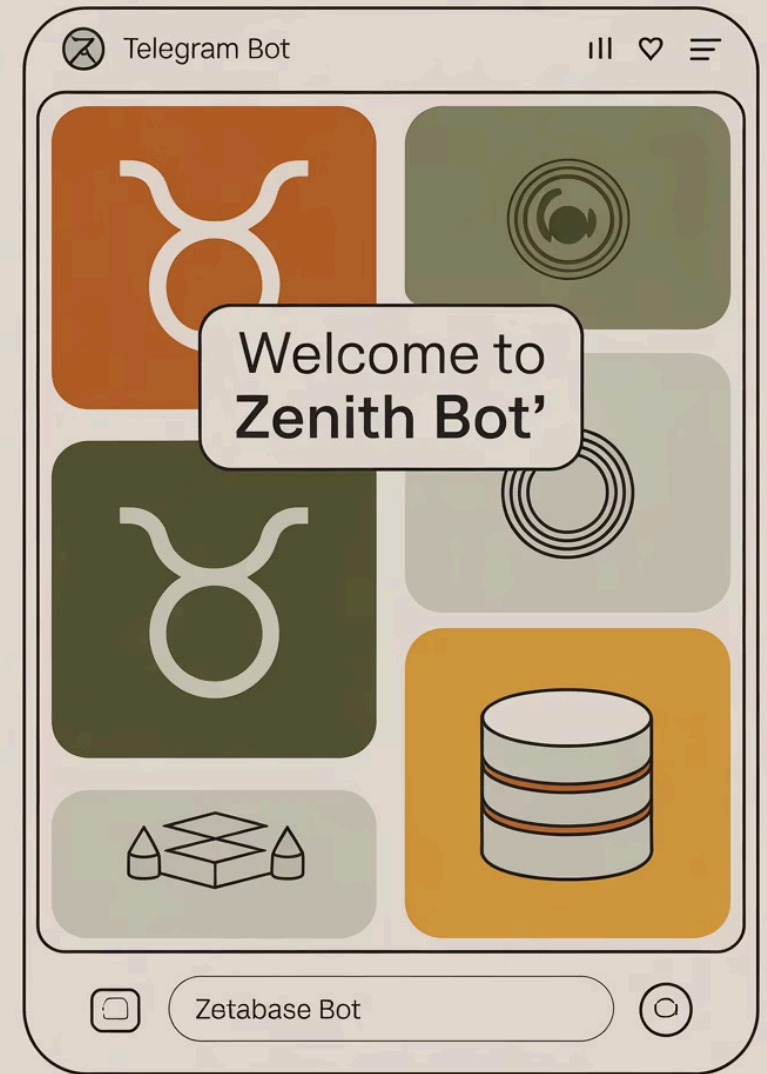


DailyZodiakBot — Telegram-бот с SQLite и ежедневными сообщениями 🤖 ✨

Изучим создание полнофункционального бота с базой данных SQLite и автоматической рассылкой гороскопов. Проект демонстрирует полный цикл разработки: от идеи до готового решения.



Что решает наш бот? 🎯

Память пользователя

Хранение настроек: знак зодиака, час уведомлений, статус подписки

Ежедневная рассылка

Автоматическое отправление персонализированных гороскопов по расписанию

Полностью локальное решение

TeleBot + SQLite + long polling — без внешних API и сложных зависимостей



Пользовательский сценарий

01

Первоначальная настройка

Команда `/start` запускает процесс выбора знака зодиака и времени получения уведомлений

02

Мгновенная проверка

Команда `/today` позволяет сразу получить гороскоп на сегодня без ожидания

03

Автоматическая доставка

Каждый день в заданное время бот присылает персональный гороскоп

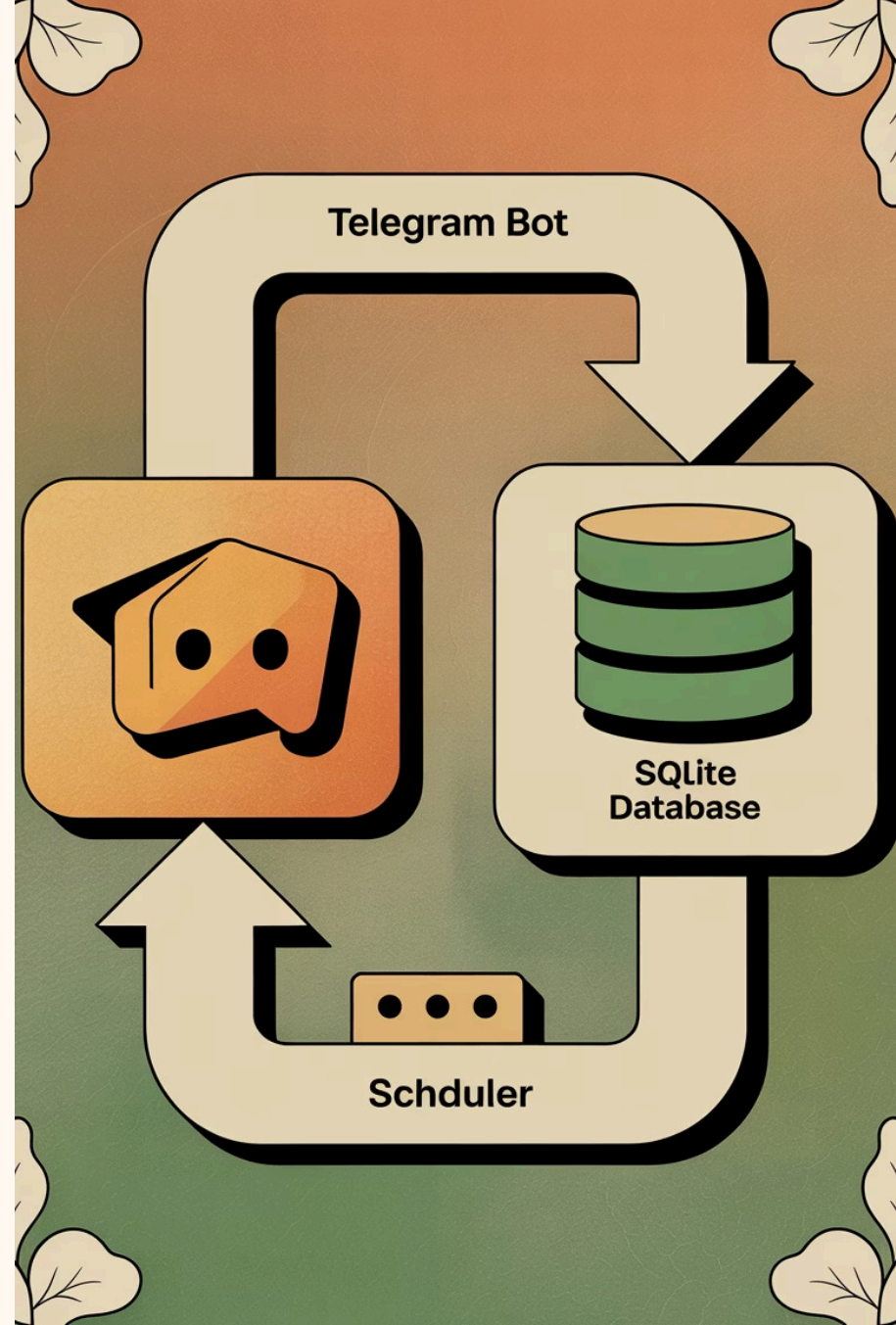
Архитектура проекта 🏗️

Основные компоненты

- TeleBot с long polling
- База данных SQLite (bot.db)
- Планировщик в отдельном потоке
- Таблица users для хранения настроек

Ключевые поля БД

- `sign` — знак зодиака
- `notify_hour` — час уведомлений
- `subscribed` — статус подписки
- `last_sent_date` — дата последней отправки



Технологический стек



Python 3.11

Современная версия Python с улучшенной производительностью и новыми возможностями



pyTelegramBotAPI

Библиотека TeleBot для взаимодействия с Telegram Bot API



SQLite3

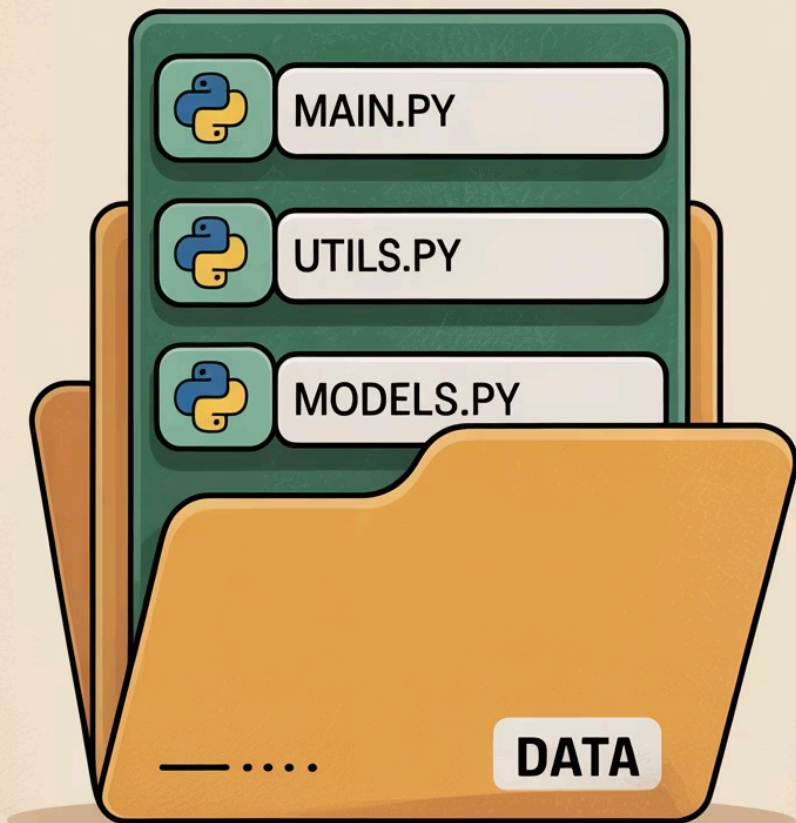
Встроенная база данных для хранения пользовательских настроек



python-dotenv

Управление переменными окружения и конфигурацией

CODE STRUCTURE



Структура проекта

config.py

Конфигурация: TOKEN, DB_PATH,
настройки логирования

db.py

Работа с SQLite: PRAGMA, CRUD
операции, инициализация БД

main.py

Основная логика: команды бота, планировщик рассылки

Безопасность: .env и .gitignore

Файл .env

```
TOKEN=your_bot_token_here  
DB_PATH=bot.db  
LOG_LEVEL=INFO
```

Все секретные данные храним в переменных окружения

Файл .gitignore

```
.env  
*.db  
__pycache__/  
*.log
```

Исключаем конфиденциальную информацию из системы контроля версий


 **Важно:** Никогда не коммитьте токены и пароли в репозиторий! Используйте .env файлы для локальной разработки.

Схема БД: таблица users

```
CREATE TABLE IF NOT EXISTS users (  
  user_id INTEGER PRIMARY KEY,  
  sign TEXT,  
  notify_hour INTEGER NOT NULL DEFAULT 9,  
  subscribed INTEGER NOT NULL DEFAULT 1,  
  last_sent_date TEXT  
);
```

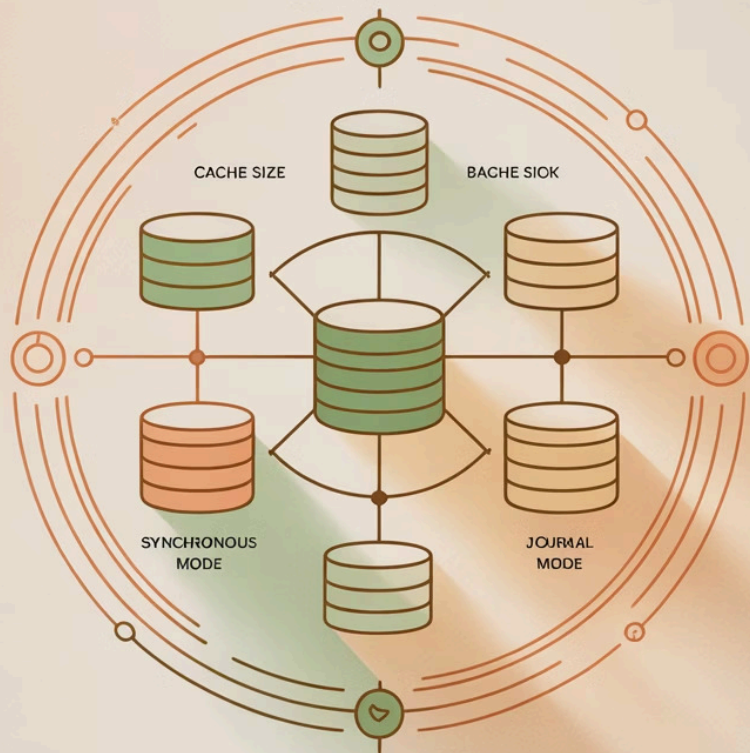
```
CREATE INDEX IF NOT EXISTS idx_users_hour  
ON users(notify_hour);
```

```
CREATE INDEX IF NOT EXISTS idx_users_sent  
ON users(last_sent_date);
```

Индексы ускоряют выборки по времени уведомлений и дате последней отправки, что критично для планировщика рассылки.

SQLITE

Connection Settings



**SQLITE CONNECTION
OPTIMIZATION SETTINGS**

Подключение к SQLite: best practices

```
conn = sqlite3.connect(DB_PATH, timeout=5.0)
conn.row_factory = sqlite3.Row
conn.execute("PRAGMA foreign_keys = ON")
conn.execute("PRAGMA journal_mode = WAL")
conn.execute("PRAGMA busy_timeout = 5000")
```

Почему WAL режим?

Write-Ahead Logging уменьшает
блокировки и ошибки "database is
locked"

Зачем Row factory?

Позволяет обращаться к колонкам по
именам вместо индексов

Инициализация БД

```
def init_db():
    """Создаём таблицы и индексы при первом запуске"""
    conn = get_connection()
    try:
        conn.execute(CREATE_USERS_TABLE)
        conn.execute(CREATE_HOUR_INDEX)
        conn.execute(CREATE_DATE_INDEX)
        conn.commit()
        logging.info("База данных инициализирована")
    except Exception as e:
        logging.error(f"Ошибка инициализации БД: {e}")
    finally:
        conn.close()
```

Функция `init_db()` выполняется при старте бота и создаёт необходимые таблицы и индексы, если они ещё не существуют. Это стандартный паттерн из курса L3.

Команды бота: полный обзор 🎮



`/start`

Регистрация нового пользователя и первоначальная настройка



`/set_sign, /set_time`

Изменение знака зодиака и времени уведомлений



`/subscribe, /unsubscribe`

Управление подпиской на ежедневные рассылки



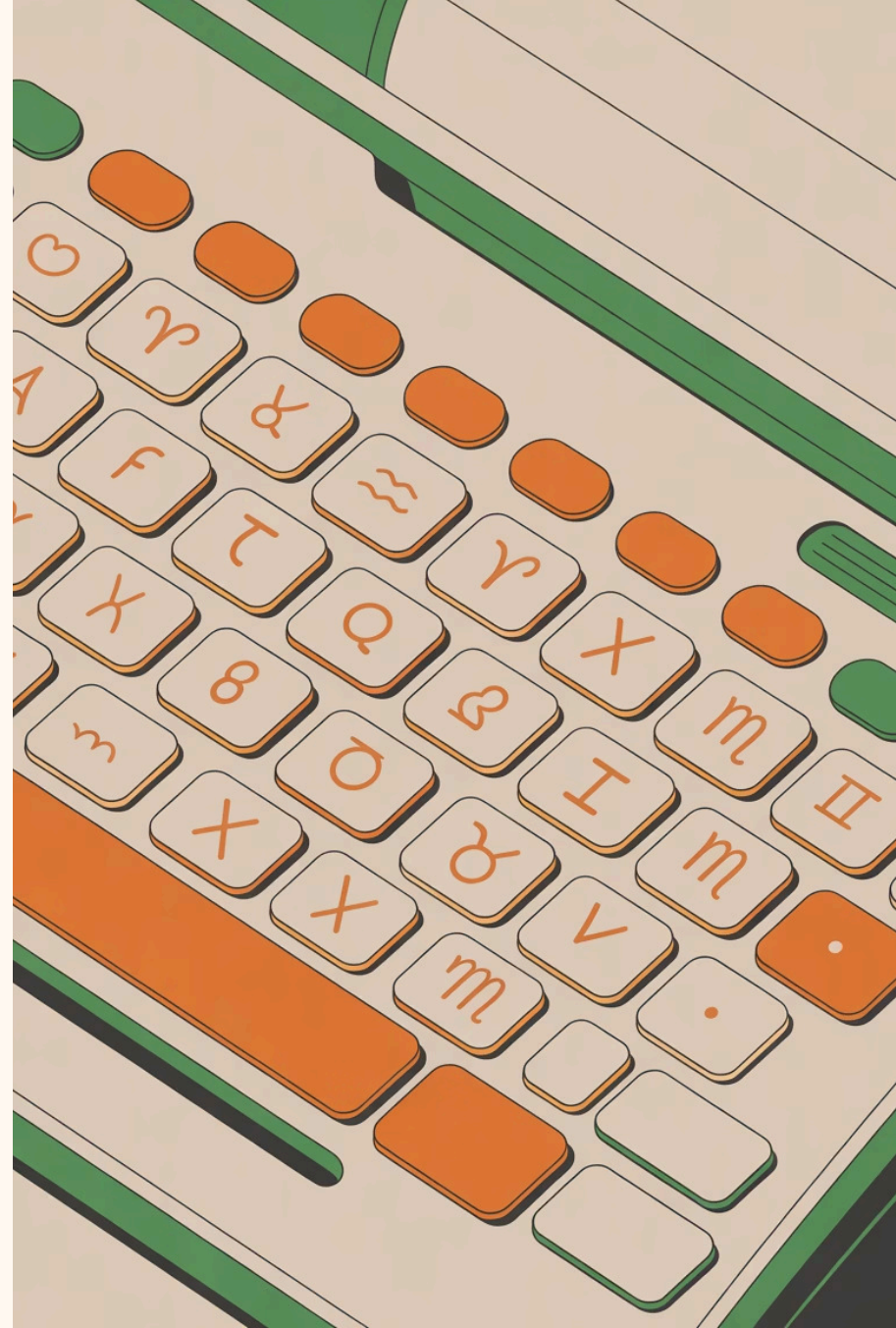
`/me, /today, /signs`

Просмотр настроек, получение гороскопа, список знаков

Все команды регистрируем через `set_my_commands` для улучшения пользовательского опыта.

Клавиатура выбора знака

```
def get_zodiac_keyboard():  
    keyboard = ReplyKeyboardMarkup(resize_keyboard=True)  
    signs = [  
        'Овен', 'Телец', 'Близнецы',  
        'Рак', 'Лев', 'Дева',  
        'Весы', 'Скорпион', 'Стрелец',  
        'Козерог', 'Водолей', 'Рыбы'  
    ]  
  
    # Разделяем знаки на ряды по 3  
    for i in range(0, len(signs), 3):  
        row_buttons = [KeyboardButton(sign) for sign in signs[i:i+3]]  
        keyboard.add(*row_buttons)  
  
    return keyboard
```



Нормализация ввода знака

Обработка текста

- Приведение к нижнему регистру
- Замена ё → е
- Поддержка алиасов (leo→лев)
- Удаление лишних символов

```
def normalize_sign(text):  
    text = text.lower().replace('ё', 'е')  
    # Убираем эмодзи и лишние пробелы  
    text = re.sub(r'^\w\s', '', text).strip()  
  
    aliases = {'leo': 'лев', 'aries': 'овен'}  
    return aliases.get(text, text)
```

После нормализации сохраняем в БД: `UPDATE users SET sign=? WHERE user_id=?;`

Настройка времени уведомлений

```
@bot.message_handler(commands=['set_time'])
def set_notification_time(message):
    try:
        parts = message.text.split()
        if len(parts) != 2:
            bot.reply_to(message, "Использование: /set_time 9")
            return

        hour = int(parts[1])
        if not (0 <= hour <= 23):
            bot.reply_to(message, "Час должен быть от 0 до 23")
            return

        update_user_hour(message.from_user.id, hour)
        bot.reply_to(message, f"Время уведомлений: {hour}:00")
    except ValueError:
        bot.reply_to(message, "Введите корректное число")
```

Валидация входных данных перед сохранением в БД — важный принцип безопасной разработки.

Управление подпиской

1

`/subscribe`

Включает ежедневную рассылку гороскопов

```
UPDATE users  
SET subscribed=1  
WHERE user_id=?;
```

2

`/unsubscribe`

Отключает автоматические уведомления

```
UPDATE users  
SET subscribed=0  
WHERE user_id=?;
```

Пользователи могут легко управлять своими предпочтениями через простые команды.

Генератор гороскопов без API 🎲

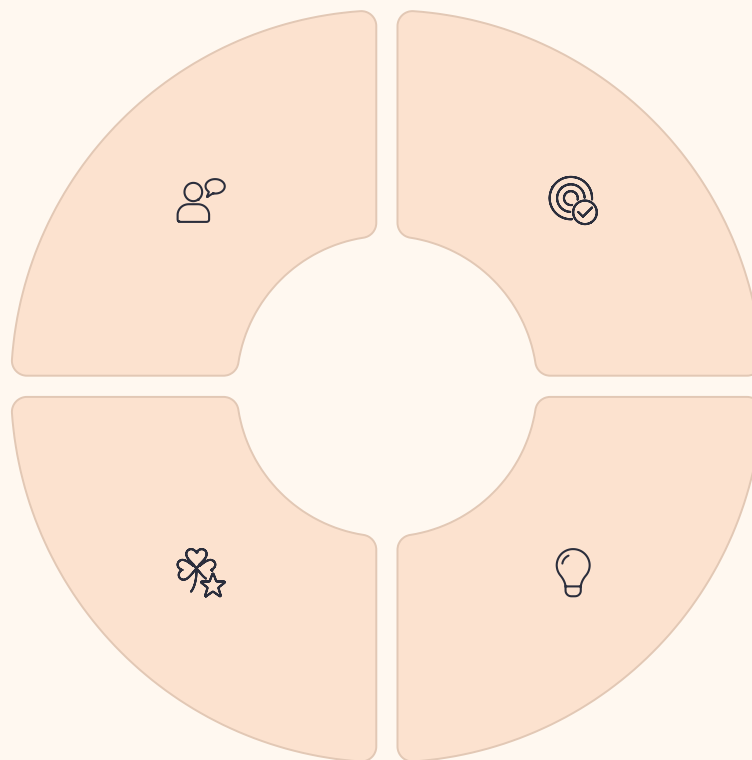
Создаём развлекательный контент без внешних зависимостей. Используем детерминированный алгоритм: хеш от (знак + дата) определяет выбор фраз из предустановленных наборов.

Интро-фразы
Приветственные обороты для начала гороскопа

Удача + цвет
Счастливое число и цвет дня

Фокус дня
Основная тема или сфера жизни

Советы
Практические рекомендации



Код генератора (упрощённый)

```
def make_daily_text(sign, date_obj):  
    # Создаём уникальный хеш для знака и даты  
    seed = f'{sign}_{date_obj.isoformat()}'  
    hash_val = int(hashlib.md5(seed.encode()).hexdigest(), 16)  
  
    # Используем хеш для выбора элементов  
    intro = INTROS[hash_val % len(INTROS)]  
    focus = FOCUSES[hash_val % len(FOCUSES)]  
    advice = ADVICES[hash_val % len(ADVICES)]  
  
    lucky_num = (hash_val % 99) + 1  
    color = COLORS[hash_val % len(COLORS)]  
  
    return f'{sign.title()}*
```



Команда /today — мгновенный тест 🚀

```
@bot.message_handler(commands=['today'])
def get_today_horoscope(message):
    user = get_user(message.from_user.id)
    if not user or not user['sign']:
        bot.reply_to(message, "Сначала выберите знак: /set_sign")
        return

    today = date.today()
    horoscope = make_daily_text(user['sign'], today)

    bot.send_message(
        message.chat.id,
        horoscope,
        parse_mode="Markdown"
    )
```

Команда позволяет протестировать генератор гороскопов без ожидания планировщика. Полезно для отладки и демонстрации функционала.

Дизайн планировщика рассылки

01

Даemon-поток

Планировщик работает в отдельном потоке-демоне, не блокирующем основной процесс

02

Проверка каждую минуту

Цикл проверяет текущий час и ищет пользователей для рассылки

03

Отметка отправки

После успешной отправки обновляем `last_sent_date` во избежание дубликатов

Простая и надёжная архитектура, подходящая для учебных проектов и небольших ботов.



SQL запрос для выбора получателей

```
SELECT user_id, sign FROM users
WHERE subscribed = 1
  AND sign IS NOT NULL
  AND notify_hour = ?
  AND (last_sent_date IS NULL
    OR last_sent_date <> ?);
```

Условия отбора

- Активная подписка
- Установлен знак зодиака
- Совпадает час уведомлений
- Сегодня ещё не отправляли

Оптимизация

Индекс по `notify_hour` значительно ускоряет выборку. Без индекса пришлось бы сканировать всю таблицу.

Отметка об успешной отправке

```
def mark_sent_today(user_id, today_str):
    """Отмечаем, что пользователю сегодня уже отправили"""
    conn = get_connection()
    try:
        cursor = conn.execute(
            "UPDATE users SET last_sent_date=? WHERE user_id=?",
            (today_str, user_id)
        )
        conn.commit()

        if cursor.rowcount == 0:
            logging.warning(f"Пользователь {user_id} не найден")
        else:
            logging.info(f"Отправка {user_id} отмечена")

    except Exception as e:
        logging.error(f"Ошибка обновления: {e}")
    finally:
        conn.close()
```

Проверка `rowcount` помогает выявить проблемы с обновлением данных.

Основной цикл планировщика

```
def scheduler_loop():
    """Основной цикл планировщика рассылки"""
    logging.info("Планировщик запущен")

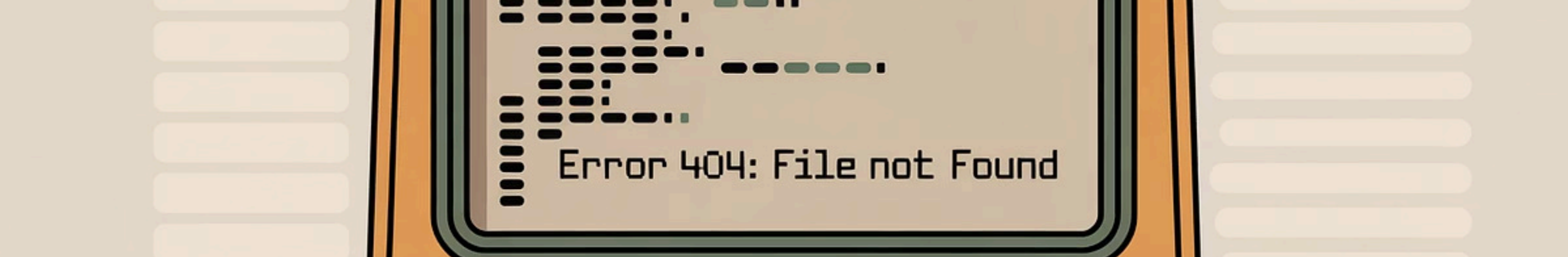
    while True:
        try:
            current_hour = datetime.now().hour
            today = date.today().isoformat()

            users = list_due_users(today, current_hour)
            logging.info(f"Найдено {len(users)} для рассылки")

            for user in users:
                horoscope = make_daily_text(user['sign'], date.today())
                bot.send_message(
                    user['user_id'],
                    horoscope,
                    parse_mode="Markdown"
                )
                mark_sent_today(user['user_id'], today)

            except Exception as e:
                logging.error(f"Ошибка планировщика: {e}")

            time.sleep(60) # Проверяем каждую минуту
```

Error 404: File not Found

Логирование и отладка

Конфигурация логгера

```
logging.basicConfig(  
    level=logging.INFO,  
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  
    handlers=[  
        logging.FileHandler('bot.log'),  
        logging.StreamHandler()  
    ]  
)
```

Что логируем

- Запуск/остановка компонентов
- Количество найденных получателей
- Ошибки отправки сообщений
- Результаты SQL операций

Качественное логирование — ключ к быстрому поиску проблем в продакшене.

Безопасность SQL: параметризация

✓ Правильно

```
cursor.execute(  
    "SELECT * FROM users WHERE user_id=?",  
    (user_id,) )
```

Параметры передаются отдельно, защищены от SQL-инъекций

✗ Неправильно

```
cursor.execute(  
    f"SELECT * FROM users WHERE user_id={user_id}"  
)
```

Строковая интерполяция создаёт уязвимости безопасности

Все SQL операции используют параметризованные запросы — это стандартный паттерн из курса L3.

Тестовые сценарии для проверки



1

Базовый флоу

1. /start → выбор знака → /set_time 9 → /me → /today

2

Тест планировщика

2. Установить час на текущий → дождаться автоотправки →
проверить last_sent_date

3

Проверка подписки

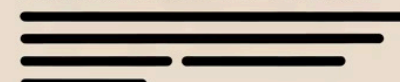
3. /unsubscribe → проверить отсутствие рассылки → /subscribe



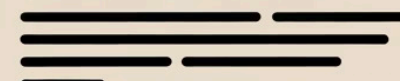
SOECKLIST



FUNCTIONAL TESTING



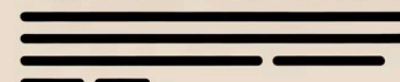
USABILITY TESTING



USABILITY TESTING



PERFORMANCE TESTING



Обработка краевых случаев

Неверный знак

Показываем подсказку со списком доступных знаков: /signs

Некорректное время

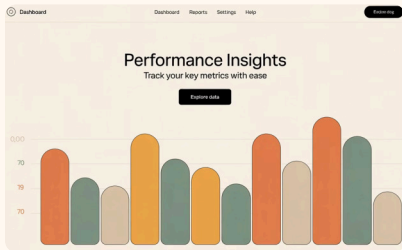
set_time -1 или 25 → валидация и отклонение с объяснением

Ошибки отправки

try/except + логирование, но планировщик продолжает работу

Предусмотреть проблемные ситуации — признак зрелого кода. Бот должен изящно обрабатывать любые пользовательские ошибки.

Возможные расширения проекта 🚀



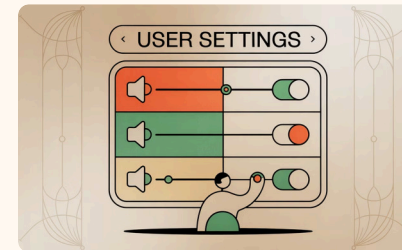
История и аналитика

/history 7 с ASCII-гистограммой активности (GROUP BY date) — применяем паттерны из курса L3



Экспорт данных

Экспорт настроек и истории в .txt файл (адаптация note_export из L3)



Персонализация

Настройка стиля сообщений, выбор тем, дополнительные виды контента

Ключевые выводы и итоги

Полноценная БД

SQLite для хранения пользовательских настроек и истории отправок

Автономный контент

Генерация развлекательного контента без внешних API и зависимостей

Надёжная рассылка

Планировщик с защитой от дубликатов и обработкой ошибок

Проверенные практики

PRAGMA/WAL/timeout, параметризация SQL, логирование

Демо и живые примеры

Что покажем

- Команда /today в действии
- Автоматическая доставка по текущему часу
- ASCII-график истории (если останется время)
- Экспорт настроек как "вау-эффект"

Важные моменты

- Акцент на простоте решения
- Без вебхуков, Docker, внешних API
- Только технологии из курса
- Reply-клавиатура удобнее новичкам





Спасибо за внимание! ✨

Вопросы?

Проект DailyZodiakBot демонстрирует полный цикл разработки Telegram-бота с базой данных. Вы изучили создание схемы БД, реализацию команд, планировщик рассылки и лучшие практики безопасности.

[Начать свой проект](#)

[Изучить код](#)

Помните: Всегда используйте `.env` для токенов и `.gitignore` для защиты конфиденциальных данных! 🔒