Geekbrains

### 

### **Создание чат-бота для напоминаний "Remind Me"**

Программа: Python-разработчик  
Яхонтова Ольга Витальевна

Москва

2024

**Содержание:**

[Введение](#_auy2dlqap99k)

[Глава 1. Теоретическая часть](#_6d1sdkpiurfz)

[1.1 Обзор аналогов и выбор подходящих технологий](#_j7voyjuv6dv6)

[1.2 Описание архитектуры и функциональности чат-бота](#_z4mmly5qawon)

[1.3 Описание функционала чат-бота](#_wzgje2sii1lq)

[Глава 2. Практическая часть](#_yjdos0hxcdl9)

[2.1 Планирование и подготовка к разработке](#_iad9tpekk1n6)

[2.2 Разработка кода чат-бота](#_bboh4d9c9grb)

[2.3 Тестирование и отладка проекта](#_kjvt5vukk7gv)

[Заключение](#_spg931fu0g4e)

[Список использованной литературы и ресурсов](#_cl66jzf0j88j)

[Приложения](#_wtt3kl5resih)

### 

### 

### **Введение**

#### **Тема проекта**

### Темой дипломного проекта является разработка чат-бота для создания и управления напоминаниями под названием "Remind Me". Чат-бот "Remind Me" предназначен для автоматизации процесса напоминаний и может использоваться в приложении Telegram, что позволяет пользователям легко и быстро настраивать напоминания через привычный интерфейс мессенджера.

#### **Цель**

### Цель проекта – создание функционального и удобного чат-бота, который позволяет пользователям задавать напоминания для важных событий и задач, управлять их расписанием и получать своевременные уведомления. Проект направлен на создание инструмента, способного улучшить организацию времени пользователя и поддерживать его продуктивность, минимизируя вероятность забывания о важных задачах.

#### **Проблема, которую решает проект**

### В повседневной жизни люди часто сталкиваются с необходимостью помнить о различных событиях, будь то рабочие задачи, встречи, важные даты или личные напоминания. Обычные приложения для напоминаний не всегда удобны, особенно если пользователю приходится постоянно переключаться между несколькими приложениями или вручную управлять списком напоминаний. Чат-бот решает эту проблему, предоставляя удобный и быстрый способ установки напоминаний прямо в мессенджере, которым пользуются ежедневно. Это снижает риск забывания, увеличивает продуктивность и освобождает время на более важные дела.

#### **Задачи проекта**

### Для достижения поставленной цели необходимо решить следующие задачи:

### Анализ аналогов и подбор технологий: Провести исследование существующих решений для напоминаний, сравнить их функциональные возможности и определить оптимальные технологии для создания чат-бота.

### Проектирование архитектуры чат-бота: Разработать архитектуру бота, включая его логику, интерфейс и хранилище данных, чтобы обеспечить надежность и удобство использования.

### Разработка основных функций:

### Создание и настройка напоминаний, включая выбор времени и даты.

### Добавление команд для редактирования и удаления напоминаний.

### Реализация различных режимов напоминаний, таких как одноразовые, ежедневные, еженедельные, и т.д.

### Настройка безопасности и конфиденциальности данных: Включить шифрование данных напоминаний и логов для защиты личной информации пользователей.

### Интеграция с Telegram API: Реализовать основные команды и взаимодействие с Telegram API для получения и отправки сообщений пользователю.

### Тестирование и отладка: Проверить корректность работы всех функций бота, провести тестирование на наличие ошибок, а также протестировать сценарии взаимодействия с пользователем.

### Документация и разработка рекомендаций для дальнейшего развития: Описать архитектуру, функции и примеры использования бота, а также предложить возможные улучшения для будущих версий.

#### **Инструменты**

### Для разработки чат-бота были выбраны следующие инструменты:

### Язык программирования Python – для написания основной логики бота и взаимодействия с Telegram API.

### Библиотека python-telegram-bot – для интеграции с Telegram API, обработки сообщений, создания команд и работы с клавиатурой.

### Библиотека cryptography – для шифрования данных напоминаний и логов с целью обеспечения конфиденциальности.

### JSON-файлы – для хранения данных о напоминаниях и их параметрах, а также для поддержания работоспособности бота в многопользовательской среде.

### Средства тестирования и отладки (ручное тестирование, логирование) – для тестирования отдельных функций и устранения ошибок.

### 

### **Основная часть**

### **Глава 1. Теоретическая часть**

#### **1.1 Обзор аналогов и выбор подходящих технологий**

##### **Обзор существующих решений**

### Для создания удобного и полезного чат-бота напоминаний важно провести анализ текущих аналогов на платформе Telegram. Рассмотрим основные популярные решения, такие как **ToDoBot** и **RemindMeBot**. Эти боты реализуют функции напоминаний, и их анализ позволит выделить сильные и слабые стороны существующих решений, что будет полезно для разработки собственного проекта.

###### 

### **ToDoBot** — это Telegram-бот для управления задачами и напоминаниями. Его функционал включает:

### Возможность создания задач и списков дел.

### Возможность устанавливать напоминания на определенное время.

### Поддержка ежедневных и еженедельных напоминаний.

### Отображение списков дел и завершенных задач.

### **Преимущества**:

### Простой и понятный интерфейс.

### Функция планирования задач и установки напоминаний.

### Поддержка повторяющихся напоминаний.

### **Недостатки**:

### Ограниченные возможности настройки напоминаний (например, нет возможности настроить напоминание на каждый второй день или задать конкретные даты).

### Отсутствие функции шифрования данных, что может быть проблемой при использовании бота для сохранения личной информации.

### **RemindMeBot** — это бот, специально разработанный для создания напоминаний. Основные возможности:

### Установка напоминаний на определенное время и дату.

### Возможность указать текст напоминания.

### Напоминания можно установить как для личных чатов, так и для групповых.

### **Преимущества**:

### Простота использования: бот быстро создает напоминания с минимальным количеством шагов.

### Интеграция с групповыми чатами, что удобно для командной работы или напоминаний в коллективе.

### **Недостатки**:

### Отсутствие повторяющихся напоминаний и гибких настроек частоты.

### Ограниченные возможности для редактирования или управления напоминаниями.

### Низкий уровень безопасности — напоминания сохраняются в незашифрованном виде.

###### **Выводы по анализу аналогов**

### На основе анализа ToDoBot и RemindMeBot можно сделать выводы о требованиях, которые необходимо реализовать в проекте. Наш бот будет сочетать удобство создания и настройки напоминаний с функциями безопасности, такими как шифрование данных, возможность удаления напоминаний, гибкие настройки частоты напоминаний (ежедневно, еженедельно, ежегодно и т.д.) и поддержка различных временных форматов.

##### **Выбор технологий и обоснование**

### На основе целей и задач проекта выбран следующий стек технологий:

### **Python** — основной язык разработки. **Обоснование**:

### Простота и популярность: Python широко используется для разработки ботов, включая Telegram-ботов, благодаря обширной экосистеме библиотек.

### Гибкость и поддержка асинхронного программирования: Python поддерживает асинхронные операции, что важно для ботов, работающих с множеством пользователей одновременно.

### Сообщество и документация: наличие богатого сообщества и множества готовых решений позволяет ускорить разработку и тестирование.

### **Библиотека *python-telegram-bot*** для взаимодействия с Telegram API. **Обоснование**:

### Широкий функционал: *python-telegram-bot* предоставляет готовые модули для взаимодействия с Telegram API, обработки сообщений, команд и inline-клавиатур.

### Асинхронные возможности: библиотека поддерживает асинхронное выполнение задач, что позволяет эффективно обрабатывать множество запросов от пользователей.

### Документированность: *python-telegram-bot* имеет качественную документацию и примеры, что упрощает внедрение новых функций и отладку.

### **Библиотека *cryptography***для шифрования данных. **Обоснование**:

### Безопасность данных: бот будет сохранять напоминания в зашифрованном формате, что обеспечит конфиденциальность пользовательских данных.

### Поддержка различных алгоритмов шифрования: *cryptography* позволяет выбирать подходящий алгоритм шифрования для разных уровней безопасности, что делает её универсальной для наших задач.

### Легкость интеграции: библиотека легко интегрируется с Python-программами и поддерживает как симметричное, так и асимметричное шифрование.

### **Формат JSON для хранения данных**. **Обоснование**:

### Простота использования и переносимости: JSON-файлы легко создавать и читать, что позволяет быстро настроить хранение напоминаний.

### Масштабируемость: при увеличении количества данных формат JSON можно заменить на базу данных SQLite, что позволит увеличить объем хранимых данных без значительных изменений в структуре проекта.

### Удобство тестирования: JSON подходит для разработки и тестирования, что особенно полезно при создании прототипов и начальной версии продукта.

### 

### **Выводы по выбору технологий**:

### Выбранный стек (Python, библиотеки *python-telegram-bot* и *cryptography*, формат JSON) позволяет создавать эффективный и безопасный чат-бот с расширенными возможностями настройки и управления напоминаниями. Python и выбранные библиотеки обеспечат быстрое взаимодействие с Telegram API и высокую гибкость в реализации функционала.

#### **1.2 Описание архитектуры и функциональности чат-бота**

##### **Архитектура бота**

### Чат-бот разработан на основе многоуровневой архитектуры, которая разделяет функциональные компоненты на слои. Это позволяет упростить развитие и поддержку бота, а также облегчает добавление новых функций и оптимизацию существующих. Основные слои архитектуры включают:

### Интерфейсный слой (Interface Layer): Этот слой отвечает за взаимодействие с пользователем через Telegram. Он обрабатывает входящие сообщения и команды, отправляет ответы пользователям, предоставляет интерфейс для выбора напоминаний и настройки параметров. На этом этапе применяется библиотека *python-telegram-bot*, которая обеспечивает работу с Telegram API, поддерживает кнопки, inline-клавиатуры, команды и текстовые сообщения.

### Слой бизнес-логики (Logic Layer): Этот слой реализует основную логику работы бота, обрабатывая данные, полученные от интерфейса, и управляя напоминаниями. Сюда входят функции для:

### Создания и управления напоминаниями.

### Проверки времени напоминаний.

### Шифрования и расшифровки данных (с использованием библиотеки *cryptography*).

### Логика включает модули для создания, редактирования, удаления и отображения напоминаний, а также управления повторяющимися напоминаниями. Это ядро функциональности бота, которое связывает интерфейс с хранилищем данных.

### Слой данных (Data Layer): Слой отвечает за сохранение и управление данными. Для хранения напоминаний и настроек пользователей используется JSON-файл, а в дальнейшем возможно расширение до использования SQLite или другой реляционной базы данных, если объем данных возрастет. Данные сохраняются в зашифрованном формате для обеспечения безопасности, благодаря библиотеке *cryptography*.

##### **Процесс обработки данных**

### Данные в боте проходят через следующие этапы:

### Получение и обработка команды. Пользователь отправляет команду, например */set\_reminder.* Интерфейсный слой передает команду в слой логики.

### Обработка логики. Слой логики выполняет операции создания, редактирования или удаления напоминания, а также проверяет валидность введенных данных.

### Сохранение данных. Данные напоминания сохраняются в JSON-файле в зашифрованном виде. Слой данных передает обратно зашифрованные данные для отправки подтверждения пользователю.

### Отправка уведомлений. Когда наступает время напоминания, бот через интерфейсный слой отправляет уведомление пользователю.

##### **Функциональные требования**

### Для обеспечения всех заявленных возможностей разработан список функциональных требований, которые определяют, что именно должен уметь делать бот:

### **Создание напоминаний**:

### Бот должен уметь создавать напоминания по заданной дате и времени.

### Напоминание может быть одноразовым или повторяющимся (ежедневным, еженедельным, ежемесячным, ежегодным).

### При создании напоминания пользователь может указать его текст и настроить частоту.

### **Просмотр списка напоминаний**:

### Пользователь должен иметь возможность просмотреть все активные напоминания с деталями, включая текст, время и частоту напоминания.

### **Удаление напоминаний**:

### Бот должен уметь удалять напоминания по запросу пользователя.

### Должна быть возможность удаления как конкретного напоминания, так и всех напоминаний.

### **Безопасность данных**:

### Все напоминания должны храниться в зашифрованном виде для обеспечения конфиденциальности данных пользователей.

##### **Нефункциональные требования**

### Для улучшения работы и устойчивости бота выделены следующие нефункциональные требования:

### **Производительность**:

### Бот должен быстро реагировать на команды, с минимальной задержкой обработки запросов.

### Эффективность работы особенно важна для операций, связанных с уведомлениями и взаимодействием с данными, чтобы не допускать задержек в отправке напоминаний.

### **Безопасность**:

### Данные, хранящиеся в боте, должны быть надежно зашифрованы, чтобы исключить несанкционированный доступ.

### Уровень защиты обеспечивает библиотека *cryptography*, которая шифрует текст напоминаний и персональные данные, включая JSON-файлы.

### **Простота использования**:

### Интерфейс должен быть интуитивно понятным и удобным даже для пользователей без технического опыта.

### Все основные функции, такие как создание, просмотр и удаление напоминаний, должны быть доступны через простые команды и кнопки.

### **1.3 Описание функционала чат-бота**

### В данном разделе подробно описаны основные функции чат-бота, реализованные для создания эффективного и удобного инструмента напоминаний. Каждая функция описана с точки зрения логики работы, используемых алгоритмов и обоснования принятых решений.

### 

### 

### **Создание напоминаний**

### Функция создания напоминания позволяет пользователю задать текст напоминания и выбрать дату и время, когда оно должно сработать. При создании напоминания бот предлагает пользователю выбрать параметры напоминания через текстовые сообщения или inline-кнопки.

### **Логика работы**:

### Пользователь отправляет команду */set\_reminder* или вводит соответствующий текст.

### Бот запрашивает выбор даты и времени напоминания.

### После ввода даты и времени бот предлагает ввести текст напоминания.

### Для удобства выбора времени используется интерактивная клавиатура, позволяющий выбрать время с интервалом 30 минут. Также возможно указать время с точностью до 1 минуты, просто напечатав его сообщением.

### В случае повторяющихся напоминаний, пользователю предлагается выбрать режим: единоразово, ежедневно, еженедельно и т.д.

### **Алгоритмы**:

### Для обработки дат и времени используется стандартная библиотека *datetime* Python.

### Настройка напоминаний осуществляется с использованием таймера, который активируется в момент, когда наступает установленное время.

### Все данные о напоминаниях сохраняются в зашифрованном виде в JSON-файле.

### **Обоснование решений**:

### Inline-кнопки и текстовые сообщения позволяют сделать процесс создания напоминания интуитивным и удобным.

### Использование интерактивной клавиатуры или отправка текстом выбора времени обеспечивает более точное и удобное взаимодействие с пользователем.

#### **Выбор режима напоминаний**

### Режимы напоминаний позволяют пользователю выбрать частоту их срабатывания. Варианты включают:

### Единоразово

### Ежедневно

### Еженедельно

### Ежемесячно

### Ежегодно

### **Логика работы**:

### После ввода режима напоминания, бот предлагает выбрать дату и время.

### Выбранный режим сохраняется как атрибут напоминания.

### При проверке напоминаний в определенное время бот анализирует выбранный режим и решает, повторять напоминание или удалять его.

### **Алгоритмы**:

### Для цикличности напоминаний реализованы условия, проверяющие текущую дату и время. В зависимости от режима, дата автоматически увеличивается на заданный интервал (например, на один день для ежедневного напоминания).

### Событие проверяется каждую минуту, чтобы учитывать временные изменения.

### **Обоснование решений**:

### Поддержка повторяющихся напоминаний делает бота полезным для пользователей с регулярными задачами, такими как ежедневные задачи или ежегодные события.

### Стандартные временные интервалы (день, неделя, месяц, год) удобны и охватывают большинство сценариев использования.

#### **Удаление напоминаний**

### Эта функция позволяют пользователям удалять существующие напоминания. Удаление необходимо для управления ненужными напоминаниями.

### **Логика работы**:

### Пользователь отправляет команду */delete\_reminder* для удаления напоминания.

### Бот отображает список напоминаний с inline-кнопками, чтобы пользователь мог выбрать нужное напоминание.

### После подтверждения бот вносит изменения и сохраняет обновленные данные в JSON-файл.

### **Алгоритмы**:

### Удаление напоминания удаляет запись из JSON-файла.

### **Обоснование решений**:

### Inline-кнопки позволяют легко выбрать напоминание для удаления без необходимости вводить текстовые команды.

#### **Шифрование данных и безопасность**

### Для безопасности данных используется шифрование. Тексты напоминаний и личные данные пользователей хранятся в зашифрованном виде.

### **Логика работы**:

### При создании напоминания текст и другие данные шифруются с использованием библиотеки *cryptography*.

### Зашифрованные данные сохраняются в JSON-файле.

### При отображении или отправке напоминания пользователю данные расшифровываются.

### **Алгоритмы**:

### Используется симметричное шифрование, что позволяет быстро и безопасно сохранять и расшифровывать данные.

### Библиотека *cryptography* предоставляет готовые методы для шифрования и расшифровки.

### **Обоснование решений**:

### Зашифрованное хранение данных предотвращает утечку конфиденциальной информации.

### Использование проверенной библиотеки *cryptography* обеспечивает высокий уровень безопасности данных, соответствующий современным стандартам.

### **Глава 2. Практическая часть**

### **2.1 Планирование и подготовка к разработке**

### В данном разделе описывается процесс планирования и подготовки к разработке чат-бота "Remind Me". Подробно раскрывается процесс интеграции библиотеки *python-telegram-bot* и других модулей, которые будут использованы для реализации функционала бота.

#### **Создание проектной документации**

### Прежде чем приступать к кодированию, была создана проектная документация, в которой определены следующие аспекты:

### Описание проекта: цель, задачи и функциональные требования.

### Технические требования: используемые технологии и библиотеки.

### Архитектура бота: описание компонентов, взаимодействие между ними.

#### **2.2 Разработка кода чат-бота**

Полный код проекта представлен в [Приложении 6](#_2la3r563xwxj).

#### **Установка библиотеки *python-telegram-bot***

### Для разработки чат-бота была выбрана популярная библиотека *python-telegram-bot*, которая предоставляет удобные методы для работы с Telegram API. Это облегчает создание и обработку команд, управление сообщениями и взаимодействие с пользователями.

### **Шаги по интеграции библиотеки**:

### Устанавливаем библиотеку с помощью команды: ``` pip install python-telegram-bot ```

### Создаем файл с расширением *.py*, который будет основным скриптом для работы с Telegram API.

### Получаем токен для бота через BotFather в Telegram. Токен необходим для аутентификации бота и отправки сообщений через Telegram.

### Настраиваем основную логику бота:

from config import TOKEN

from telegram import Update, InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardMarkup

from telegram.ext import Application, CommandHandler, ContextTypes, CallbackQueryHandler, MessageHandler, filters

# Читаем TOKEN из config

…

# Обработчик команды /start

async def start(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text(

'Привет! 👋\n'

'Я твой верный бот для создания напоминаний. 🕰\n\n'

'Ты когда-нибудь забывал сделать что-то важное? 🤔\n'

'Не беда! Я помогу тебе помнить обо всех делах, даже о тех, которые так хочется забыть. 😅\n\n'

'Вот что я умею:\n'

'📌 Хочешь добавить напоминание?\n'

'Просто используй команду: /set\_reminder\n'

'📋 Посмотреть все свои напоминания? Легко!\n'

'Жми: /list\_reminders\n'

'🗑️ Удалить лишнее напоминание? Без проблем!\n'

'Введи: /delete\_reminder\n\n'

'Доверь мне свои заботы, а сам наслаждайся жизнью! 😎'

)

…

def main():

application = Application.builder().token(TOKEN).build()

application.add\_handler(CommandHandler('start', start))

…

loop = asyncio.get\_event\_loop()

loop.create\_task(run\_scheduler(application))

application.run\_polling()

if \_\_name\_\_ == '\_\_main\_\_':

main()

### Этот код запускает бота, который отвечает на команду */start*.

#### **Интеграция библиотеки *cryptography***

### Для обеспечения безопасности данных, таких как тексты напоминаний и личные данные пользователей, была выбрана библиотека *cryptography*. Она позволяет эффективно шифровать и расшифровывать данные, что важно для защиты конфиденциальной информации.

### **Шаги по интеграции библиотеки**:

### Установите библиотеку с помощью команды: ``` pip install cryptography ```

### В коде бота добавьте функцию загрузки ключа шифрования для шифрования и расшифровки данных:

def load\_key():

if os.path.exists('secret.key'):

with open('secret.key', 'rb') as key\_file:

return key\_file.read()

else:

key = Fernet.generate\_key()

with open('secret.key', 'wb') as key\_file:

key\_file.write(key)

return key

### В этом коде используется симметричное шифрование для защиты данных, что гарантирует, что только авторизованный пользователь может расшифровать и просматривать свои напоминания.

**Интеграция библиотеки *json***

Для хранения данных в проекте был выбран формат JSON. Он является удобным для записи структурированных данных в текстовый файл, а также легко читаемым человеком. Библиотека *json* в Python позволяет работать с данными в этом формате, обеспечивая простоту сериализации (преобразования объектов Python в строку) и десериализации (восстановления объектов из строки).

### **Шаги по интеграции библиотеки**:

##### Установка и использование библиотеки *json*

##### Библиотека *json* входит в стандартную библиотеку Python, поэтому установка не требуется. Для работы с JSON используется два основных метода:

* *json.dump()* — для записи Python-объекта в файл в формате JSON.
* *json.load()* — для чтения данных из файла и преобразования их в Python-объект.

1. Для использования этих методов в коде бота необходимо добавить функции для записи и чтения из JSON-файла:

import json

DATA\_FILE = 'reminders.json'

# Словарь для хранения временных данных пользователя

user\_data = {}

# Инициализация данных

if not os.path.exists(DATA\_FILE):

with open(DATA\_FILE, 'w') as f:

json.dump({}, f)

# Функция для загрузки напоминаний

def load\_reminders() -> dict:

try:

with open(DATA\_FILE, 'r', encoding='utf-8') as file:

content = file.read()

if content:

return json.loads(content)

return {}

except (FileNotFoundError, json.JSONDecodeError) as e:

logger.error(f"Error loading reminders: {e}")

return {}

# Функция для сохранения напоминаний

def save\_reminders(reminders: dict):

with open(DATA\_FILE, 'w', encoding='utf-8') as file:

json.dump(reminders, file, ensure\_ascii=False, indent=4)

1. В этом коде мы создаем структуру данных, которая содержит напоминания пользователя, и сохраняем ее в файл с расширением *.json*. Затем мы читаем этот файл и восстанавливаем данные в формате Python.

##### **Хранение данных напоминаний**

##### В проекте данные напоминаний хранятся в файле JSON. Каждый пользователь может иметь несколько напоминаний, которые будут сериализованы в JSON-формат. Таким образом, для добавления и удаления напоминаний используется следующий алгоритм:

### Для добавления нового напоминания мы извлекаем текущие данные из файла JSON, добавляем новое напоминание и сохраняем обновленные данные обратно в файл.

### Для удаления напоминания извлекаем данные, фильтруем нужное напоминание по его идентификатору или времени, затем обновляем файл.

### Функции для добавления и удаления напоминаний:

…

# Функция для добавления напоминания

def add\_reminder(user\_id: str, time: str, message: str, repeat: str):

reminders = load\_reminders()

if user\_id not in reminders:

reminders[user\_id] = []

encrypted\_message = cipher.encrypt(message.encode()).decode() # Шифруем сообщение

reminders[user\_id].append({

'time': time,

'message': encrypted\_message,

'repeat': repeat,

'sent': False

})

logger.info(f"Added encrypted reminder for user {user\_id}: {encrypted\_message}")

save\_reminders(reminders)

…

# Обработчик команды /delete\_reminder

async def delete\_reminder(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user\_id = str(update.message.from\_user.id)

reminders = load\_reminders()

user\_reminders = reminders.get(user\_id, [])

if not user\_reminders:

await update.message.reply\_text('У вас нет напоминаний для удаления.')

return

keyboard = []

for i, reminder in enumerate(user\_reminders, 1):

decrypted\_message = cipher.decrypt(reminder['message'].encode()).decode()

reminder\_text = f"{reminder['time']} - {decrypted\_message}"

keyboard.append([InlineKeyboardButton(reminder\_text, callback\_data=f"delete\_{i}")])

keyboard.append([InlineKeyboardButton("Отмена", callback\_data="cancel")])

reply\_markup = InlineKeyboardMarkup(keyboard)

await update.message.reply\_text('Выберите напоминание для удаления:', reply\_markup=reply\_markup)

### Пример хранения данных в JSON-файле с применением шифрования:

{

"456194055": [

{

"time": "2024-11-08 13:30",

"message": "gAAAAABnKJ1L4GkA0jBCv4\_Jz-J638Y0jPrFJO4-MDNUvTPeNUkngJCasKFrvIWF1Ig1HbSR6Z2jqQw3EYT6HWc\_VggbDN4fOZgDL1brhwvFZeXwgIovCOrfD6Wo6OgRBPWzU\_FlWQ2BeFetMMxh\_G8B1uLJvIhxwg==",

"repeat": "ежедневно",

"sent": false

},

{

"time": "2024-11-11 13:30",

"message": "gAAAAABnKJ-h2tmWG6MihMKwD8Y8BtzjTEFJBnSJmf37axDs-dPqlBq5XvKur565QqT6Akq0EJMEr3gLXa6auhnWjGsCD2EmdlgK8i6Nj0ck2hEQquRg8slVEkE5WYNXna9SWhCPIsiEvEpKWOziYEFhNAVplUscZQ==",

"repeat": "еженедельно",

"sent": false

}

],

"205960794": [],

"5233739009": []

}

#### **Логирование**

Логирование является важной частью разработки, так как позволяет отслеживать работу приложения, фиксировать ошибки и важные события. В Python для этого используется стандартная библиотека *logging*, которая предоставляет гибкие средства для записи логов с разными уровнями важности.

##### **1. Работы с библиотекой *logging***

В первую очередь настраиваем конфигурацию логирования, указав, куда будут записываться логи, какой уровень важности записей использовать и формат сообщений.

import logging

log\_filename = 'remind\_me\_bot.log'

logging.basicConfig(

filename=log\_filename,

filemode='a',

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

level=logging.INFO,

encoding='utf-8'

)

logger = logging.getLogger(\_\_name\_\_)

Пример функции в коде, где используется логирование:

def add\_reminder(user\_id: str, time: str, message: str, repeat: str):

reminders = load\_reminders()

if user\_id not in reminders:

reminders[user\_id] = []

encrypted\_message = cipher.encrypt(message.encode()).decode() # Шифруем сообщение

reminders[user\_id].append({

'time': time,

'message': encrypted\_message,

'repeat': repeat,

'sent': False

})

logger.info(f"Added encrypted reminder for user {user\_id}: {encrypted\_message}")

save\_reminders(reminders)

Пример лог файла:

2024-11-07 15:01:08,627 - \_\_main\_\_ - INFO - Added encrypted reminder for user *456194055*: gAAAAABnLKwELquRqXF8iWwXgVs6l\_ll\_FEGAk9mkslJOxwyqSbeF1aMbzErmRu47p-5nO-YnvHYCJyRYgmnJrsWWsiiqjyUGw==

2024-11-07 15:01:08,630 - \_\_main\_\_ - INFO - Reminder set by user *456194055*: gAAAAABnLKwEU6NLX6QRJ9IZdmktB4Pm1drIWZmOOAyJ05NgXeiiMwzAUKDsCHtLcHbljh-Rc48bgpdZBl-HpW4wBkHpNtRC8A==

В процессе планирования и подготовки к разработке был проделан важный этап выбора технологий для написания кода. Для хранения данных был использован формат JSON, что позволяет легко сохранять и читать напоминания пользователей. Работа с библиотекой *json* обеспечила простоту и гибкость в управлении данными. Логирование было настроено с использованием стандартной библиотеки *logging*, что обеспечило прозрачность работы бота и возможность оперативного выявления ошибок и проблем.

### **2.3 Тестирование и отладка проекта**

#### **Процесс тестирования**

Тестирование проекта "Remind Me" включало несколько этапов, на которых проверялись все основные функции чат-бота. Основным методом тестирования был ручной ввод данных в интерфейсе чат-бота, что позволяло детально проверять работу всех функций, таких как создание, просмотр и удаление напоминаний.

Тестирование проводилось в несколько этапов:

1. Тестирование создания напоминаний:
   * На этом этапе было проверено, как бот реагирует на запросы пользователя для создания новых напоминаний. Пользователь мог отправить команду для создания напоминания, и бот должен был корректно отреагировать на введенные данные (текст напоминания, время, режим повторения).
   * Были использованы различные варианты временных интервалов, чтобы убедиться в правильности работы механизмов для единичных и повторяющихся напоминаний.
   * Каждое созданное напоминание сохранялось в структуре данных, и проверялось, что оно отображается корректно в списке напоминаний пользователя.

Пример команды для создания напоминаний представлен в [Приложении 1](#_jm819m7loum).

1. Тестирование просмотра напоминаний:
   * Проверялась функция отображения списка текущих напоминаний пользователя. После того как напоминания были добавлены в систему, пользователю предоставлялась возможность запросить список всех своих напоминаний.
   * В ходе тестирования было важно проверить корректность вывода информации, отображения времени напоминаний в правильном формате и их текстового содержания.
   * Также тестировалась обработка пустых списков напоминаний (когда у пользователя нет созданных напоминаний).

Пример команды для просмотра напоминаний представлен в [Приложении 2](#_5v7iorufelu).

1. Тестирование удаления напоминаний:
   * На этом этапе проверялась возможность удаления напоминаний, созданных пользователем. Бот должен был корректно удалять выбранные напоминания, при этом убедившись, что они исчезают из списка и больше не отображаются.

Пример команды для удаления напоминания представлен в [Приложении 3](#_c34l854ripvu).

1. Тестирование шифрования данных:
   * Важной частью тестирования было также проверка работы системы шифрования данных. Бот использует библиотеку *cryptography* для шифрования текста напоминаний перед их сохранением и для расшифровки при отображении. Были проведены тесты на проверку корректности процесса шифрования и расшифровки.
   * Для этого были созданы несколько напоминаний, зашифрованные данные записывались в файл JSON, и позже проверялась возможность их корректной расшифровки.
   * Проверялось, что зашифрованные данные не доступны без соответствующего ключа и что расшифровка данных происходит корректно.

Пример зашифрованных данных в JSON-файле представлен в [Приложении 4](#_ne7h11l1589p).

1. Тестирование обработки ошибок:
   * Важной частью тестирования было проверка устойчивости бота к ошибкам и исключениям. Для этого были проверены такие случаи, как:
     + Невалидный формат времени (например, некорректная дата или время).
     + Попытка добавить напоминание без обязательных полей (например, без текста или времени).
   * В этих случаях бот должен был корректно обрабатывать ошибки, информировать пользователя о некорректных действиях и не допускать сбоев в работе.

Пример тестирования обработки ошибок представлен в [Приложении 5](#_v4qel9sydln1).

#### **Ручное тестирование в чате с ботом**

Тестирование было проведено в реальной среде взаимодействия с ботом. Были приглашены пользователи для взаимодействия с ботом. Каждый этап тестирования проводился в реальном времени, и по мере получения результатов вносились необходимые корректировки в код.

1. В процессе тестирования использовался разнообразный набор команд, который позволял проверять все функции бота. Каждый шаг был документирован, а результаты записывались для дальнейшего анализа и оптимизации.
2. После каждого теста проводились исправления в функциональности или интерфейсе, если возникали проблемы с определенными командами или логикой работы.

#### 

#### **Выводы по тестированию**

1. **Положительные результаты**: Все основные функции бота, включая создание, удаление напоминаний, а также просмотр и обработка ошибок, работали стабильно. Бот корректно обрабатывал пользовательские запросы, и данные сохранялись без ошибок.
2. **Обнаруженные проблемы**: В ходе тестирования были обнаружены мелкие баги, такие как некорректное отображение времени для некоторых форматов ввода времени.
3. **Решения проблем**: Все проблемы были решены путем изменения логики обработки данных, улучшения алгоритмов валидации и добавления дополнительных проверок на корректность ввода данных. В результате были значительно улучшены стабильность работы бота и пользовательский интерфейс.

Таким образом, ручное тестирование в реальных условиях позволило выявить и устранить проблемы, обеспечив надежную и стабильную работу бота "Remind Me".

### 

### **Заключение**

В ходе выполнения дипломного проекта был разработан функциональный чат-бот "Remind Me", ориентированный на помощь пользователям в управлении их задачами и напоминаниями. Основной целью работы было создание удобного, безопасного и надежного инструмента для своевременного напоминания о важных событиях и задачах, с возможностью использования гибких режимов напоминаний и персонализированного интерфейса. Данная цель была достигнута: реализованный бот соответствует заявленным требованиям, а также обладает рядом уникальных возможностей, повышающих его удобство и ценность для пользователей.

#### **Итоги проведенной работы**

В ходе проекта были выполнены все основные этапы, начиная от анализа аналогов и выбора подходящих технологий, до разработки и тестирования готового решения. В ходе реализации были достигнуты следующие результаты:

1. Создана удобная система управления напоминаниями с возможностью гибкого выбора режимов повторения, включая единоразовые, ежедневные, еженедельные и другие виды напоминаний.
2. Реализована функция шифрования данных, обеспечивающая безопасность хранения личной информации пользователя. Напоминания сохраняются в зашифрованном виде, что исключает возможность их несанкционированного доступа.
3. Разработан дружелюбный интерфейс с поддержкой текстовых команд, а также с возможностью выбора времени через интерактивную клавиатуру или вводом времени текстом. Это делает взаимодействие с ботом интуитивно понятным и простым.
4. Проведено тестирование и отладка всех функций, что позволило устранить обнаруженные ошибки и оптимизировать работу приложения.

#### **Важность и польза созданного чат-бота для пользователей**

Созданный чат-бот имеет большую практическую ценность для пользователей, которым важно управлять своим временем и задачами. Он позволяет организовать напоминания и отслеживать выполнение задач в удобном формате, что особенно актуально в условиях плотного графика и большого количества обязательств. Доступность бота через Telegram делает его полезным инструментом для широкой аудитории, так как Telegram используется на разных платформах, что обеспечивает высокий уровень доступности.

#### **Возможные направления для дальнейшего развития**

Чат-бот "Remind Me" обладает широким потенциалом для дальнейшего развития. В рамках будущих улучшений можно рассмотреть следующие направления:

1. Расширение функционала:
   * Добавление новых режимов напоминаний, таких как напоминания по геолокации или на основе погодных условий.
   * Введение поддержки напоминаний с вложениями, например, возможность прикреплять фотографии, документы или ссылки.
   * Реализация функции "умных напоминаний", которые автоматически подстраиваются под распорядок пользователя, используя машинное обучение для анализа его предпочтений и частоты взаимодействий с ботом.
2. Поддержка других платформ:
   * Адаптация бота для работы на других мессенджерах, таких как WhatsApp или Viber, чтобы расширить аудиторию и предоставить доступ к функции управления напоминаниями пользователям других платформ.
   * Создание веб-версии или мобильного приложения для пользователей, которые предпочитают использовать напоминания вне мессенджеров.
3. Улучшение безопасности:
   * Внедрение двухфакторной аутентификации для повышения защиты данных пользователей.
   * Использование более продвинутых алгоритмов шифрования для усиления безопасности конфиденциальной информации.
   * Добавление контроля доступа к данным через дополнительные уровни аутентификации при взаимодействии с ботом.
4. Оптимизация работы:
   * Улучшение алгоритмов обработки команд для ускорения реакции бота и минимизации задержек при взаимодействии.
   * Оптимизация использования ресурсов сервера и добавление поддержки горизонтального масштабирования для повышения стабильности работы при увеличении числа пользователей.
   * Проведение дополнительной аналитики и оптимизации интерфейса для упрощения взаимодействия и сокращения времени на выполнение основных операций.

#### **Выводы**

Выполненный дипломный проект демонстрирует возможности применения современных технологий для создания полезного инструмента, который может помочь пользователям эффективнее управлять своим временем. В процессе разработки удалось реализовать все запланированные функции, а также обеспечить удобство и безопасность работы бота. В перспективе "Remind Me" может стать еще более универсальным и полезным инструментом, если будут реализованы предложенные направления для дальнейшего развития.

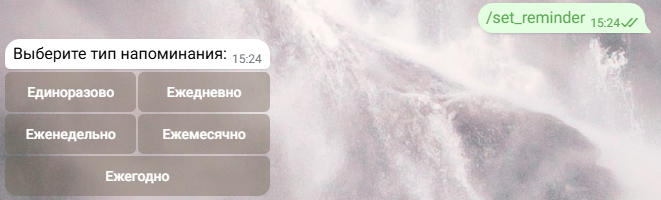
### 

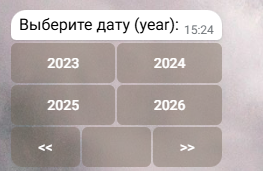
### **Список использованной литературы и ресурсов**

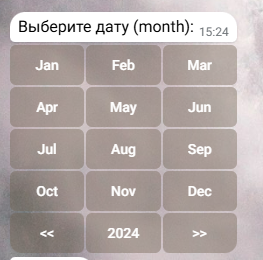
1. **Документация и справочные материалы**:
   * [Python Documentation](https://docs.python.org/3/) — официальный сайт документации Python, содержащий полное описание синтаксиса и библиотек языка. Использовался для написания кода, отладки и тестирования проекта.
   * [python-telegram-bot Documentation](https://python-telegram-bot.readthedocs.io/) — официальная документация библиотеки *python-telegram-bot*, с описанием функций и примеров кода для создания и управления Telegram-ботами. Данный ресурс служил основой для работы с Telegram API.
   * [Cryptography Library Documentation](https://cryptography.io/en/latest/) — официальный сайт библиотеки *cryptography*, описывающий алгоритмы шифрования и механизмы защиты данных. Использовался для реализации шифрования данных напоминаний, сохраненных в JSON-файлах.
2. **Руководства и книги**:
   * [Martelli, A., Ravenscroft, A., & Holden, S. (2013). *Python Cookbook: Recipes for Mastering Python 3*. O'Reilly Media](https://elhacker.info/manuales/OReilly%204%20GB%20Collection/O'Reilly%20-%20Python%20Cookbook.pdf) — книга с примерами и рекомендациями по разработке на Python, которая была полезна при создании архитектуры и написании кода для бота.
3. **Статьи и обучающие материалы**:
   * [*How to Build a Telegram Bot Using Python* (2022)](https://www.freecodecamp.org/news/how-to-create-a-telegram-bot-using-python/). FreeCodeCamp — пошаговое руководство по созданию Telegram-ботов с использованием Python и библиотеки python-telegram-bot, использовалось для настройки базовых функций бота.
4. **Видеоматериалы и курсы**:
   * GeekBrains: Курс Python-разработчик — курс, пройденный автором, в рамках которого были получены знания по основам программирования на Python.

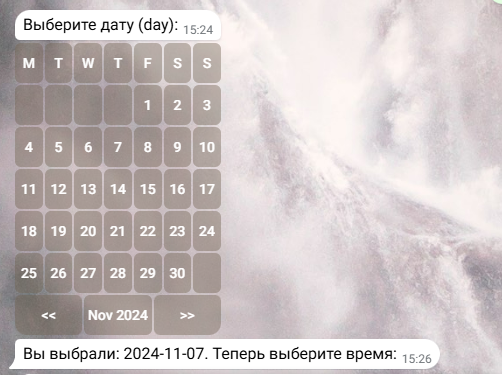
### **Приложения**

#### **Приложение 1.** Тестирование создания напоминания

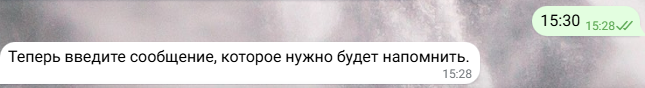






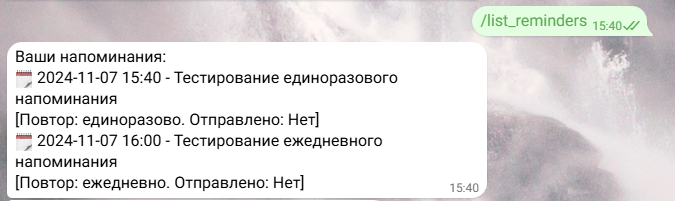






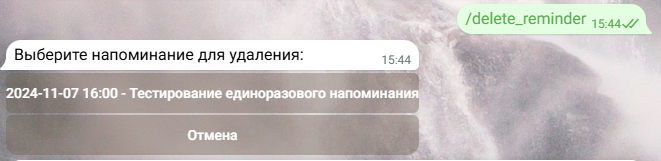


#### **Приложение 2.** Тестирование просмотра напоминаний.

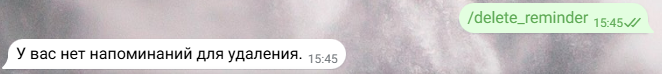




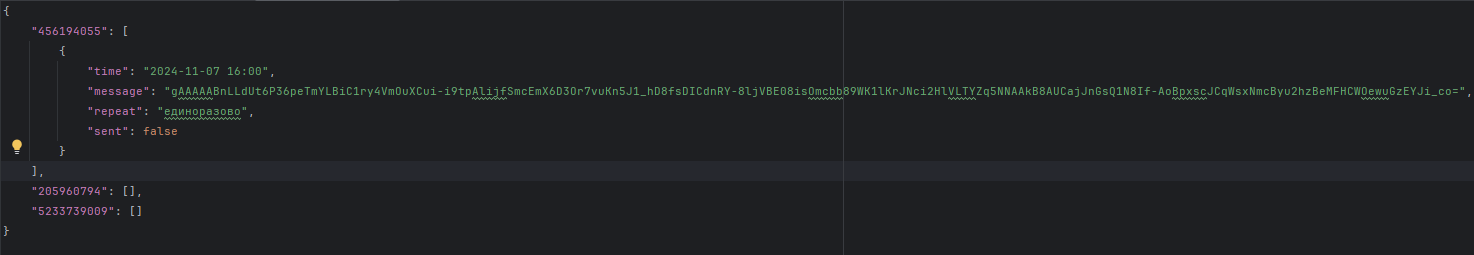
#### **Приложение 3.** Тестирование удаления напоминаний.

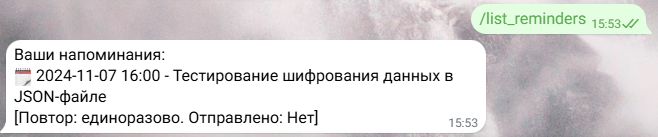




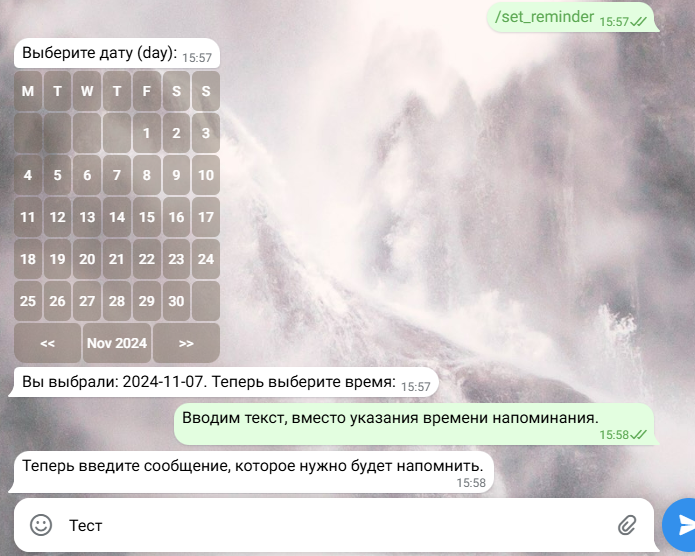


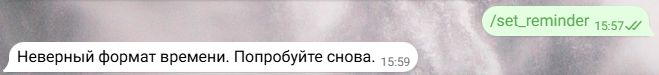
#### **Приложение 4.** Тестирование шифрования данных.





#### **Приложение 5.** Тестирование обработки ошибок





#### **Приложение 6.** Полный код чат-бота.

import json

import os

import logging

from datetime import datetime, timedelta

from telegram import Update, InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardMarkup

from telegram.ext import Application, CommandHandler, ContextTypes, CallbackQueryHandler, MessageHandler, filters

from apscheduler.schedulers.asyncio import AsyncIOScheduler

from apscheduler.triggers.interval import IntervalTrigger

from telegram\_bot\_calendar import DetailedTelegramCalendar, LSTEP

from dateutil.relativedelta import relativedelta

import asyncio

from config import TOKEN

from cryptography.fernet import Fernet

# Читаем TOKEN из config

DATA\_FILE = 'reminders.json'

# Функция для загрузки ключа шифрования

def load\_key():

if os.path.exists('secret.key'):

with open('secret.key', 'rb') as key\_file:

return key\_file.read()

else:

key = Fernet.generate\_key()

with open('secret.key', 'wb') as key\_file:

key\_file.write(key)

return key

KEY = load\_key() # Загружаем ключ

cipher = Fernet(KEY)

# Словарь для хранения временных данных пользователя

user\_data = {}

# Добавим словарь для хранения ID сообщений

user\_messages = {}

# Настройка логирования

log\_filename = 'remind\_me\_bot.log'

logging.basicConfig(

filename=log\_filename,

filemode='a',

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

level=logging.INFO,

encoding='utf-8'

)

logger = logging.getLogger(\_\_name\_\_)

# Инициализация данных

if not os.path.exists(DATA\_FILE):

with open(DATA\_FILE, 'w') as f:

json.dump({}, f)

# Функция для загрузки напоминаний

def load\_reminders() -> dict:

try:

with open(DATA\_FILE, 'r', encoding='utf-8') as file:

content = file.read()

if content:

return json.loads(content)

return {}

except (FileNotFoundError, json.JSONDecodeError) as e:

logger.error(f"Error loading reminders: {e}")

return {}

# Функция для сохранения напоминаний

def save\_reminders(reminders: dict):

with open(DATA\_FILE, 'w', encoding='utf-8') as file:

json.dump(reminders, file, ensure\_ascii=False, indent=4)

# Функция для добавления напоминания

def add\_reminder(user\_id: str, time: str, message: str, repeat: str):

reminders = load\_reminders()

if user\_id not in reminders:

reminders[user\_id] = []

encrypted\_message = cipher.encrypt(message.encode()).decode() # Шифруем сообщение

reminders[user\_id].append({

'time': time,

'message': encrypted\_message,

'repeat': repeat,

'sent': False

})

logger.info(f"Added encrypted reminder for user {user\_id}: {encrypted\_message}")

save\_reminders(reminders)

# Функция для создания клавиатуры выбора типа напоминания

def get\_type\_keyboard():

keyboard = [

[

InlineKeyboardButton("Единоразово", callback\_data='единоразово'),

InlineKeyboardButton("Ежедневно", callback\_data='ежедневно')

],

[

InlineKeyboardButton("Еженедельно", callback\_data='еженедельно'),

InlineKeyboardButton("Ежемесячно", callback\_data='ежемесячно')

],

[

InlineKeyboardButton("Ежегодно", callback\_data='ежегодно')

]

]

return InlineKeyboardMarkup(keyboard)

# Функция для создания клавиатуры выбора времени с интервалом 30 минут

def get\_time\_keyboard():

hours = [f'{i:02}' for i in range(0, 24)] # Часы от 00 до 23

minutes = [f'00', f'30'] # Минуты с интервалом 30 минут

time\_keyboard = [[f'{hour}:{minute}' for minute in minutes] for hour in hours]

return ReplyKeyboardMarkup(time\_keyboard, one\_time\_keyboard=True)

# Функция для удаления сообщений по ID

async def delete\_user\_messages(user\_id: str, context: ContextTypes.DEFAULT\_TYPE):

if user\_id in user\_messages:

for message\_id in user\_messages[user\_id]:

try:

await context.bot.delete\_message(chat\_id=user\_id, message\_id=message\_id)

except Exception as e:

logger.error(f"Failed to delete message for user {user\_id}: {e}")

user\_messages[user\_id] = []

# Обработчик команды /start

async def start(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text(

'Привет! 👋\n'

'Я твой верный бот для создания напоминаний. 🕰\n\n'

'Ты когда-нибудь забывал сделать что-то важное? 🤔\n'

'Не беда! Я помогу тебе помнить обо всех делах, даже о тех, которые так хочется забыть. 😅\n\n'

'Вот что я умею:\n'

'📌 Хочешь добавить напоминание?\n'

'Просто используй команду: /set\_reminder\n'

'📋 Посмотреть все свои напоминания? Легко!\n'

'Жми: /list\_reminders\n'

'🗑️ Удалить лишнее напоминание? Без проблем!\n'

'Введи: /delete\_reminder\n\n'

'Доверь мне свои заботы, а сам наслаждайся жизнью! 😎'

)

# Обработчик команды /set\_reminder

async def set\_reminder(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user\_id = str(update.message.from\_user.id)

user\_data[user\_id] = {}

msg = await update.message.reply\_text(

'Выберите тип напоминания:',

reply\_markup=get\_type\_keyboard()

)

user\_messages[user\_id] = [msg.message\_id] # Сохраняем ID сообщения

# Обработчик нажатий на кнопки выбора типа напоминания

async def button\_handler(update: Update, context: ContextTypes.DEFAULT\_TYPE):

query = update.callback\_query

user\_id = str(query.from\_user.id)

# Получаем выбранный тип напоминания

reminder\_type = query.data

user\_data[user\_id]['type'] = reminder\_type

# Запускаем календарь для выбора даты

calendar, step = DetailedTelegramCalendar().build()

await query.answer()

msg = await query.edit\_message\_text(f"Выберите дату ({LSTEP[step]}):", reply\_markup=calendar)

user\_messages[user\_id].append(msg.message\_id) # Добавляем ID нового сообщения

# Обработчик выбора даты

async def calendar\_handler(update: Update, context: ContextTypes.DEFAULT\_TYPE):

query = update.callback\_query

user\_id = str(query.from\_user.id)

# Проверка наличия данных для пользователя

if user\_id not in user\_data:

await query.answer()

await query.edit\_message\_text('Произошла ошибка. Попробуйте снова.')

return

# Работа с календарем

result, key, step = DetailedTelegramCalendar().process(query.data)

if not result and key:

msg = await query.edit\_message\_text(f"Выберите дату ({LSTEP[step]}):", reply\_markup=key)

user\_messages[user\_id].append(msg.message\_id)

elif result:

user\_data[user\_id]['date'] = result

await query.answer()

# Отправляем новое сообщение для выбора времени

msg = await context.bot.send\_message(chat\_id=query.message.chat\_id,

text=f"Вы выбрали: {result}. Теперь выберите время:",

reply\_markup=get\_time\_keyboard())

user\_messages[user\_id].append(msg.message\_id)

# Обработчик выбора времени и получения сообщения

async def handle\_message(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user\_id = str(update.message.from\_user.id)

if user\_id not in user\_data or 'date' not in user\_data[user\_id]:

await update.message.reply\_text('Сначала выберите тип напоминания с помощью команды /set\_reminder.')

return

# Проверяем, выбрано ли время

if 'time' not in user\_data[user\_id]:

user\_data[user\_id]['time'] = update.message.text

user\_messages[user\_id].append(update.message.message\_id) # Сохраняем ID сообщения пользователя о времени

msg = await update.message.reply\_text('Теперь введите сообщение, которое нужно будет напомнить.')

user\_messages[user\_id].append(msg.message\_id) # Сохраняем ID сообщения с просьбой ввести текст

else:

reminder\_message = update.message.text

reminder\_date = user\_data[user\_id]['date']

reminder\_time = user\_data[user\_id]['time']

reminder\_type = user\_data[user\_id]['type']

# Добавляем напоминание

reminder\_datetime\_str = f"{reminder\_date} {reminder\_time}"

try:

reminder\_datetime = datetime.strptime(reminder\_datetime\_str, '%Y-%m-%d %H:%M')

encrypted\_message = cipher.encrypt(reminder\_message.encode()).decode()

add\_reminder(user\_id, reminder\_datetime.strftime('%Y-%m-%d %H:%M'), reminder\_message, reminder\_type)

logger.info(f"Reminder set by user {user\_id}: {encrypted\_message}")

await update.message.reply\_text(f'Напоминание добавлено на {reminder\_datetime\_str}. '

f'Повтор: {reminder\_type}')

except ValueError:

await update.message.reply\_text('Неверный формат времени. Попробуйте снова.')

# Сохраняем ID сообщения пользователя с текстом напоминания

user\_messages[user\_id].append(update.message.message\_id)

# Удаляем временные данные и сообщения пользователя

await delete\_user\_messages(user\_id, context)

del user\_data[user\_id]

# Обработчик команды /list\_reminders

async def list\_reminders(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user\_id = str(update.message.from\_user.id)

reminders = load\_reminders()

user\_reminders = reminders.get(user\_id, [])

if not user\_reminders:

await update.message.reply\_text('У вас нет установленных напоминаний.')

return

message\_lines = ['Ваши напоминания:']

for i, reminder in enumerate(user\_reminders):

decrypted\_message = cipher.decrypt(reminder['message'].encode()).decode() # Дешифруем сообщение

message\_lines.append(

f"🗓️ {reminder['time']} - {decrypted\_message}\n"

f"[Повтор: {reminder['repeat']}. Отправлено: {'Да' if reminder['sent'] else 'Нет'}]"

)

await update.message.reply\_text('\n'.join(message\_lines))

# Обработчик команды /delete\_reminder

async def delete\_reminder(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user\_id = str(update.message.from\_user.id)

reminders = load\_reminders()

user\_reminders = reminders.get(user\_id, [])

if not user\_reminders:

await update.message.reply\_text('У вас нет напоминаний для удаления.')

return

keyboard = []

for i, reminder in enumerate(user\_reminders, 1):

decrypted\_message = cipher.decrypt(reminder['message'].encode()).decode()

reminder\_text = f"{reminder['time']} - {decrypted\_message}"

keyboard.append([InlineKeyboardButton(reminder\_text, callback\_data=f"delete\_{i}")])

keyboard.append([InlineKeyboardButton("Отмена", callback\_data="cancel")])

reply\_markup = InlineKeyboardMarkup(keyboard)

await update.message.reply\_text('Выберите напоминание для удаления:', reply\_markup=reply\_markup)

# Обработчик выбора напоминания для удаления

async def delete\_button\_handler(update: Update, context: ContextTypes.DEFAULT\_TYPE):

query = update.callback\_query

user\_id = str(query.from\_user.id)

data = query.data

reminders = load\_reminders()

user\_reminders = reminders.get(user\_id, [])

if data.startswith('delete\_'):

# Исправляем индекс, чтобы он соответствовал индексации списка (отнимаем 1)

index = int(data.split('\_')[1]) - 1

if 0 <= index < len(user\_reminders):

deleted\_reminder = user\_reminders.pop(index)

# Сохраняем обновления, не затрагивая напоминания других пользователей

reminders[user\_id] = user\_reminders

save\_reminders(reminders)

await query.answer()

await query.edit\_message\_text(f'Напоминание '

f'"{cipher.decrypt(deleted\_reminder["message"].encode()).decode()}"'

f' удалено.')

logger.info(f"Reminder deleted by user {user\_id}: "

f"{cipher.encrypt(deleted\_reminder['message'].encode()).decode()}")

else:

await query.answer()

await query.edit\_message\_text('Ошибка: Неверный номер напоминания.')

logger.error(f"Failed to delete reminder for user {user\_id}: Invalid index {index}")

elif data == 'cancel':

await query.answer()

await query.edit\_message\_text('Удаление напоминания отменено.')

# Функция для проверки и отправки напоминаний

async def check\_reminders(application: Application):

now = datetime.now()

now\_str = now.strftime('%Y-%m-%d %H:%M')

logger.info(f"Checking reminders at {now\_str}")

reminders = load\_reminders()

updated\_reminders = {}

for user\_id, user\_reminders in reminders.items():

new\_reminders = []

for reminder in user\_reminders:

reminder\_time = datetime.strptime(reminder['time'], '%Y-%m-%d %H:%M')

if reminder\_time <= now and not reminder['sent']:

chat\_id = int(user\_id)

try:

# Отправляем напоминание

decrypted\_message = cipher.decrypt(reminder['message'].encode()).decode()

await application.bot.send\_message(chat\_id=chat\_id, text=decrypted\_message)

logger.info(f"Sent message to chat\_id {chat\_id}: "

f"{cipher.encrypt(decrypted\_message.encode()).decode()}")

reminder['sent'] = True

# Добавляем одноразовое повторное напоминание через 10 минут

if not reminder.get('is\_repeat', False):

new\_reminder\_time = now + timedelta(minutes=10)

new\_reminders.append({

'time': new\_reminder\_time.strftime('%Y-%m-%d %H:%M'),

'message': cipher.encrypt(decrypted\_message.encode()).decode(),

'repeat': 'единоразово',

'sent': False,

'is\_repeat': True

})

save\_reminders(reminders)

except Exception as e:

logger.error(f"Failed to send message to chat\_id {chat\_id}: {e}")

# Добавляем повторное напоминание, если оно не одноразовое

if reminder['repeat'] != 'единоразово':

if reminder['repeat'] == 'ежедневно':

reminder\_time += timedelta(days=1)

elif reminder['repeat'] == 'еженедельно':

reminder\_time += timedelta(weeks=1)

elif reminder['repeat'] == 'ежемесячно':

reminder\_time += relativedelta(months=1)

elif reminder['repeat'] == 'ежегодно':

reminder\_time += relativedelta(years=1)

reminder['time'] = reminder\_time.strftime('%Y-%m-%d %H:%M')

reminder['sent'] = False

new\_reminders.append(reminder)

else:

new\_reminders.append(reminder)

updated\_reminders[user\_id] = new\_reminders

if updated\_reminders != reminders:

save\_reminders(updated\_reminders)

# Функция для запуска планировщика

async def run\_scheduler(application: Application):

scheduler = AsyncIOScheduler()

scheduler.add\_job(check\_reminders, IntervalTrigger(minutes=1), args=[application])

scheduler.start()

while True:

await asyncio.sleep(1)

def main():

application = Application.builder().token(TOKEN).build()

application.add\_handler(CommandHandler('start', start))

application.add\_handler(CommandHandler('set\_reminder', set\_reminder))

application.add\_handler(CommandHandler('list\_reminders', list\_reminders))

application.add\_handler(CommandHandler('delete\_reminder', delete\_reminder))

application.add\_handler(CallbackQueryHandler(delete\_button\_handler, pattern='^(delete\_.\*|cancel)$'))

application.add\_handler(CallbackQueryHandler(button\_handler,

pattern='^(единоразово|ежедневно|еженедельно|ежемесячно|ежегодно)$'))

application.add\_handler(CallbackQueryHandler(calendar\_handler))

application.add\_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle\_message))

loop = asyncio.get\_event\_loop()

loop.create\_task(run\_scheduler(application))

application.run\_polling()

if \_\_name\_\_ == '\_\_main\_\_':

main()