МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Техническое руководство по созданию Telegram-Бота для интерактивного музея

по проектной практике

Выполнил:
студент учебной группы 241-351, Чуфаров С.Б.

Москва, 2025

Содержание

Введение

В современном мире автоматизация и цифровизация делают взаимодействие с посетителями более удобным и эффективным. Telegram — одна из самых популярных платформ для обмена сообщениями благодаря своей открытости, API и поддержке ботов.

Создавая бота для Telegram, вы получаете: Возможность взаимодействия с широкой аудиторией; Простоту интеграции с различными системами; Возможность использовать богатый функционал (кнопки, меню, inline-режимы). Это делает Telegram отличной платформой для реализации информационных систем в сфере культуры и туризма.

Данное руководство предназначено для разработчиков, начинающих или опытных, кто хочет создать такой бот с нуля. В нем подробно разобраны архитектура системы, этапы разработки, используемые технологии и инструменты, а также визуализированы процессы с помощью UML-диаграмм и схем.

Общая архитектура системы

Компоненты с-мы

Перед началом разработки важно понять основные компоненты системы и их взаимодействие. Это поможет структурировать работу и избежать ошибок на поздних этапах.

Таблица. 1. Компоненты с-мы

Компонент	Назначение	Технологии
Пользователь	Посетитель музея, взаимодействующий с ботом	Телеграмм-клиент (мобильное приложение или десктоп)
	Обеспечивает обмен сообщениями между пользователем и сервером	Встроенный API Telegram
Бэкенд (сервер)	Обработка логики работы бота, формирование ответов, управление данными	Python (с использованием библиотек pyTelegramBotAPI, aiogram или python-telegram-bot)
База данных / Хранилище	Хранение информации о маршрутах, выставках, пользователях	SQLite (легко внедряется), PostgreSQL (более масштабируемо), JSON-файлы

Эта архитектура — классическая модель клиент-серверных приложений: пользователь взаимодействует через интерфейс (Telegram), сообщения проходят через API платформы к вашему серверу (бэкенду), который обрабатывает запросы и обращается к базе данных за необходимой информацией.

Визуализация архитектуры

Это разделение позволяет легко масштабировать систему: например, при росте количества пользователей можно расширять базу данных или переносить бэкенд на облачные сервисы.

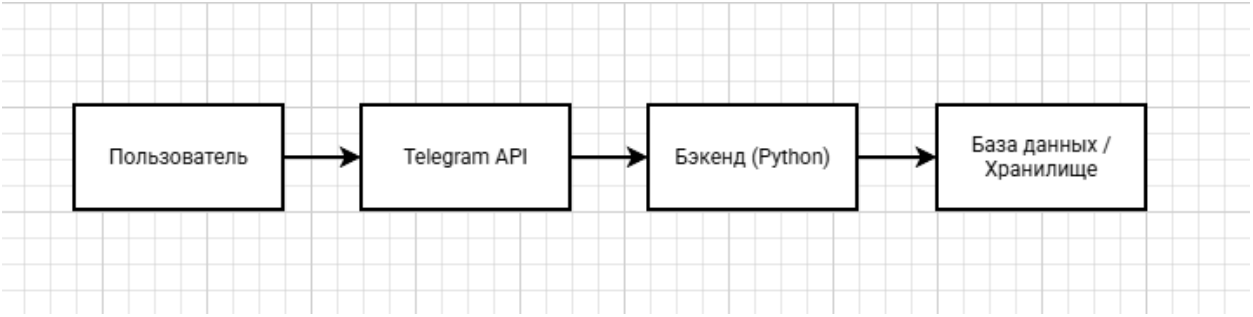


Рис.1. UML Диаграмма компонентов

Процессы взаимодействия

Для лучшего понимания разберем сценарий: пользователь запускает бота командой /start, получает меню и выбирает активность.

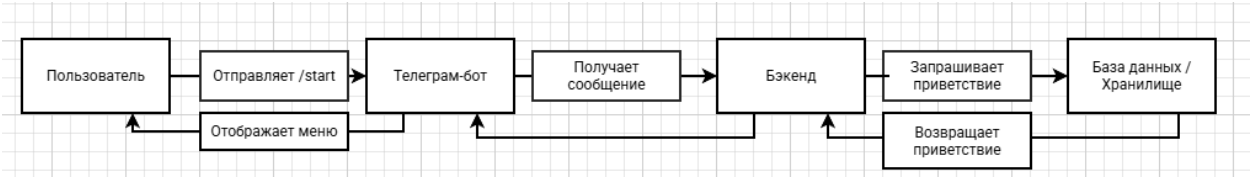


Рис. 2. UML Диаграмма последовательности

Это помогает понять порядок действий: пользователь инициирует команду; бот получает сообщение; обращается к серверу; сервер запрашивает данные у базы; возвращает ответ — все происходит очень быстро.

Процесс разработки — пошаговая инструкция

Шаг 1: подготовка окружения

1.1 Установите Python последней версии.

1.2 Установить необходимые библиотеки: “pip install pyTelegramBotAPI”

Шаг 2: регистрация бота в Telegram

2.1 Откройте Telegram.

2.2 Найдите бота @BotFather.

2.3 Создайте нового бота командой /newbot.

2.4 Получите токен API — он понадобится для авторизации вашего бота.

Шаг 3: проектирование структуры данных

3.1 Решите, как будете хранить информацию о маршрутах:

В виде JSON-файлов? Тогда проще для небольшого проекта.

В базе данных? Для более сложных сценариев лучше использовать SQLite или PostgreSQL.

В данном руководстве рассматривается создание бота без использования базы данных.

Шаг 4: реализация основных модулей

4.1 Создайте файл app.py. В нем подключите библиотеку для работы с Telegram API:

```
TOKEN = 'ВАШ_ТОКЕН_ЗДЕСЬ'
bot = telebot.TeleBot(TOKEN)

@bot.message_handler(commands=['start'])
def handle_start(message):
    # Отправляем приветствие и меню
    bot.send_message(message.chat.id, "Добро пожаловать! Выберите маршрут:", reply_markup=menu()

def menu():
    # Создаем клавиатуру с кнопками маршрутов
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.add('Историческая экспозиция', 'Современное искусство')
    return markup

@bot.message_handler(func=lambda message: True)
def handle_message(message):
    if message.text == 'Историческая экспозиция':
        # Выводим описание маршрута из файла или базы данных
        bot.send_message(message.chat.id, "Описание маршрута Историческая экспозиция...")
    elif message.text == 'Современное искусство':
        bot.send_message(message.chat.id, "Описание маршрута Современное искусство...")
    else:
        bot.send_message(message.chat.id, "Пожалуйста, выберите один из вариантов.")

if __name__ == '__main__':
    bot.polling()
```

Рис. 3. Базовый код

Шаг 5: тестирование и отладка

5.1 Запустить бота локально

5.2 Проверить работу команд и интерактивных элементов

Шаг 6: деплой на сервер (по желанию)

6.1 Для постоянной работы рекомендуется разместить бота на облачном сервере или VPS:

Используйте сервисы вроде Heroku или DigitalOcean.

6.2 Настройте автоматический запуск скрипта.

6.3 Обеспечьте безопасность хранения токена (например, через переменные окружения).

Примечание: В проекте рассматривается создание бота без использования базы данных.

Модификации

В ходе разработки был реализован ряд значимых изменений в базовый код бота для повышения его функциональности и удобства использования:

Добавление меню сайта с inline-кнопками: Введены inline-кнопки ("Главная", "О проекте", "Журнал", "Ресурсы", "Участники"), которые открывают соответствующие страницы сайта. Это позволяет пользователю быстро перейти к нужным разделам без необходимости запоминать URL или вводить команды вручную.

Интерактивное меню экспонатов: Создано отдельное меню для выбора конкретных экспонатов ("Экспонат 1", "Экспонат 2", "Экспонат 3"). При выборе пользователь получает описание выбранного экспоната, а также фотографию. Это делает ознакомление более наглядным и интересным.

Функция возврата в главное меню: Введена функция get_return_to_main(), которая создает клавиатуру с кнопкой "Вернуться в меню". Используется после просмотра информации или контактов для удобной навигации.

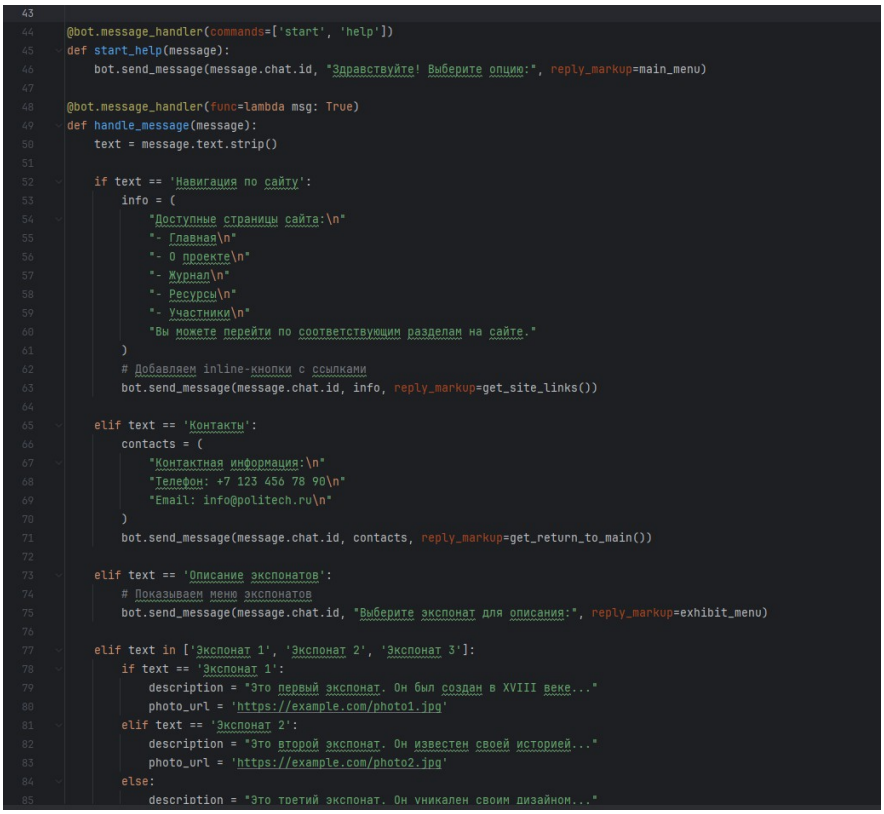
Обработка дополнительных команд: Добавлены реакции на команды "Позвонить" и "Написать письмо" — бот предоставляет контактную информацию. Введена обработка команды "История экспоната" и раздела "Фотографии" — расширяет возможности получения информации.

Обработка неизвестных сообщений: В случае неподдерживаемых команд или сообщений бот выводит сообщение с предложением выбрать из меню или использовать /help.

Таким образом, итоговый вариант проекта выглядит так:

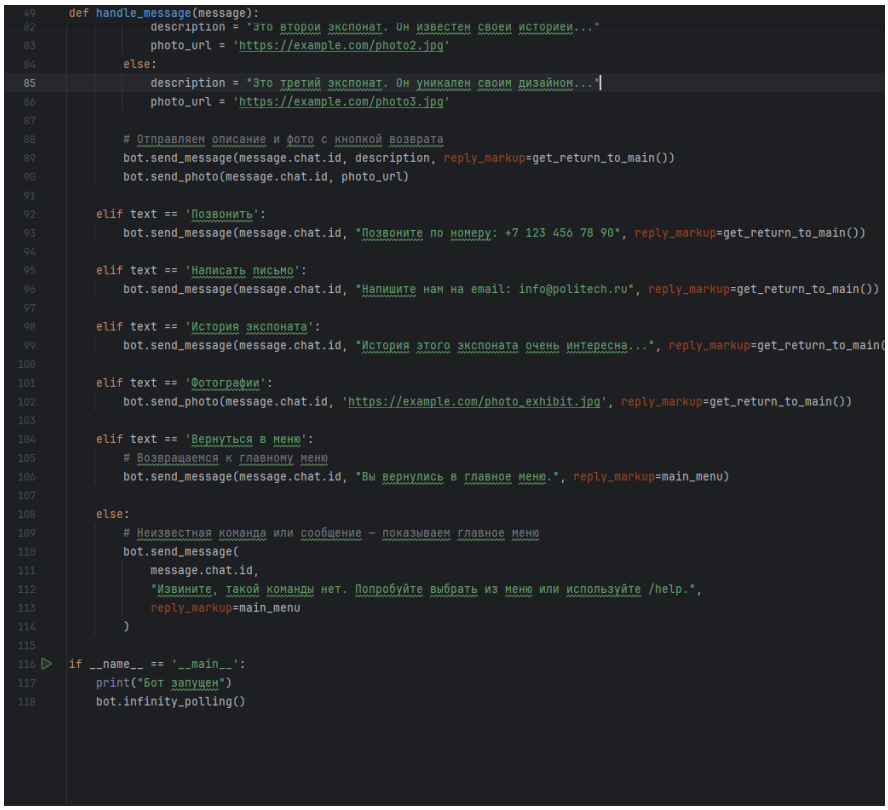
```python
import os
import telebot
from dotenv import load_dotenv

load_dotenv('tocken.env')  # ваш файл с токеном

BOT_TOKEN = os.environ.get('BOT_TOKEN')
bot = telebot.TeleBot(BOT_TOKEN)

# Основное меню
main_menu = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
main_menu.row('Навигация по сайту', 'Контакты')
main_menu.row('Описание экспонатов')

# Меню экспонатов
exhibit_menu = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
exhibit_menu.row('Экспонат 1', 'Экспонат 2')
exhibit_menu.row('Экспонат 3')

# Функция для клавиатуры "Вернуться в меню"
def get_return_to_main():  6 usages
    markup = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Вернуться в меню')
    return markup

# Функция для inline-кнопок "Перейти на сайт" и "Подробнее"
def get_site_links():  1 usage
    markup = telebot.types.InlineKeyboardMarkup()
    # Основные разделы сайта
    markup.add(
        telebot.types.InlineKeyboardButton("Главная", url="index.html"),
        telebot.types.InlineKeyboardButton("О проекте", url="about.html")
    )
    markup.add(
        telebot.types.InlineKeyboardButton("Журнал", url="journal.html"),
        telebot.types.InlineKeyboardButton("Ресурсы", url="resources.html")
    )
    markup.add(
        telebot.types.InlineKeyboardButton("Участники", url="participants.html")
    )
    # Подробные описания страниц (можно сделать отдельными кнопками или сообщениями)
    return markup
```

Рис.4 Часть кода 1/3



```python

@bot.message_handler(commands=['start', 'help'])
def start_help(message):
    bot.send_message(message.chat.id, "Здравствуйте! Выберите опцию:", reply_markup=main_menu)

@bot.message_handler(func=lambda msg: True)
def handle_message(message):
    text = message.text.strip()

    if text == 'Навигация по сайту':
        info = (
            "Доступные страницы сайта:\n"
            "- Главная\n"
            "- О проекте\n"
            "- Журнал\n"
            "- Ресурсы\n"
            "- Участники\n"
            "Вы можете перейти по соответствующим разделам на сайте."
        )
        # Добавляем inline-кнопки с ссылками
        bot.send_message(message.chat.id, info, reply_markup=get_site_links())

    elif text == 'Контакты':
        contacts = (
            "Контактная информация:\n"
            "Телефон: +7 123 456 78 90\n"
            "Email: info@politech.ru\n"
        )
        bot.send_message(message.chat.id, contacts, reply_markup=get_return_to_main())

    elif text == 'Описание экспонатов':
        # Показываем меню экспонатов
        bot.send_message(message.chat.id, "Выберите экспонат для описания:", reply_markup=exhibit_menu)

    elif text in ['Экспонат 1', 'Экспонат 2', 'Экспонат 3']:
        if text == 'Экспонат 1':
            description = "Это первый экспонат. Он был создан в XVIII веке..."
            photo_url = 'https://example.com/photo1.jpg'
        elif text == 'Экспонат 2':
            description = "Это второй экспонат. Он известен своей историей..."
            photo_url = 'https://example.com/photo2.jpg'
        else:
            description = "Это третий экспонат. Он уникален своим дизайном..."
```

Рис.5 Часть кода 2/3



```python
def handle_message(message):
            description = "Это второй экспонат. Он известен своей историей..."
            photo_url = 'https://example.com/photo2.jpg'
        else:
            description = "Это третий экспонат. Он уникален своим дизайном..."
            photo_url = 'https://example.com/photo3.jpg'

        # Отправляем описание и фото с кнопкой возврата
        bot.send_message(message.chat.id, description, reply_markup=get_return_to_main())
        bot.send_photo(message.chat.id, photo_url)

    elif text == 'Позвонить':
        bot.send_message(message.chat.id, "Позвоните по номеру: +7 123 456 78 90", reply_markup=get_return_to_main())

    elif text == 'Написать письмо':
        bot.send_message(message.chat.id, "Напишите нам на email: info@politech.ru", reply_markup=get_return_to_main())

    elif text == 'История экспоната':
        bot.send_message(message.chat.id, "История этого экспоната очень интересна...", reply_markup=get_return_to_main(

    elif text == 'Фотографии':
        bot.send_photo(message.chat.id, 'https://example.com/photo_exhibit.jpg', reply_markup=get_return_to_main())

    elif text == 'Вернуться в меню':
        # Возвращаемся к главному меню
        bot.send_message(message.chat.id, "Вы вернулись в главное меню.", reply_markup=main_menu)

    else:
        # Неизвестная команда или сообщение — показываем главное меню
        bot.send_message(
            message.chat.id,
            "Извините, такой команды нет. Попробуйте выбрать из меню или используйте /help.",
            reply_markup=main_menu
        )

if __name__ == '__main__':
    print("Бот запущен")
    bot.infinity_polling()
```

Рис.6 Часть кода 3/3

Заключение

Создание Telegram-бота — это увлекательный процесс объединения программирования, дизайна интерфейсов и проектирования систем. Это руководство охватывает все основные этапы создания Telegram-бота — от проектирования архитектуры до реализации и тестирования. Использование UML диаграмм помогает визуализировать структуру системы и процессы взаимодействия.

Реализованные модификации значительно расширили возможности бота: добавлены новые разделы информации, улучшена навигация внутри диалогов, внедрены интерактивные элементы (inline-кнопки), что сделало взаимодействие более удобным и современным. Такой подход способствует повышению вовлеченности пользователя, облегчает получение информации и делает взаимодействие более комфортным.