1. **Структура проекта**

telegram\_subtitle\_bot/

├── src/

│ ├── \_\_init\_\_.py

│ ├── bot/

│ │ ├── \_\_init\_\_.py

│ │ ├── handlers.py

│ │ ├── commands.py

│ │ └── middleware.py

│ ├── services/

│ │ ├── \_\_init\_\_.py

│ │ ├── ai\_service.py

│ │ ├── subtitle\_parser.py

│ │ └── template\_manager.py

│ ├── utils/

│ │ ├── \_\_init\_\_.py

│ │ ├── file\_handler.py

│ │ ├── logger.py

│ │ └── config.py

│ └── models/

│ ├── \_\_init\_\_.py

│ └── montage\_sheet.py

├── templates/

│ ├── montage\_sheet\_template.txt

│ └── example\_montage\_sheet.txt

├── tests/

│ ├── \_\_init\_\_.py

│ ├── test\_bot.py

│ ├── test\_ai\_service.py

│ └── test\_subtitle\_parser.py

├── .env

├── requirements.txt

├── main.py

├── README.md

└── config.yaml

1. **Контекст и цель проекта**

Создай полнофункциональный Telegram-бот на Python для автоматической обработки субтитров и генерации монтажных листов с использованием искусственного интеллекта. Бот должен принимать файлы субтитров от пользователей, обрабатывать их с помощью OpenAI API и возвращать структурированный монтажный лист на основе заранее загруженного шаблона.

ТРЕБОВАНИЯ К АРХИТЕКТУРЕ:

- Использовать асинхронное программирование с python-telegram-bot

- Применить паттерн MVC для разделения логики

- Обеспечить надежную обработку ошибок и логгирование

- Поддержать форматы субтитров: SRT, VTT

- Интегрировать с OpenAI GPT-4 для анализа контента

- Обеспечить конфигурацию через переменные окружения

1. **Основной файл main.py**

# Создай файл main.py - точку входа в приложение Telegram-бота

# ТРЕБОВАНИЯ:

# 1. Импортировать все необходимые модули для async работы

# 2. Настроить логгирование с ротацией файлов

# 3. Загрузить конфигурацию из .env файла

# 4. Инициализировать Telegram Application с обработчиками

# 5. Добавить graceful shutdown с обработкой сигналов

# 6. Включить middleware для rate limiting

# 7. Добавить обработку исключений на уровне приложения

"""

Telegram Bot для обработки субтитров и генерации монтажных листов.

Основной модуль запуска бота с полной конфигурацией и обработкой ошибок.

Author: AI Assistant

Version: 1.0.0

"""

import asyncio

import logging

import os

import signal

from pathlib import Path

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

from dotenv import load\_dotenv

from src.bot.handlers import BotHandlers

from src.bot.commands import BotCommands

from src.bot.middleware import RateLimitingMiddleware

from src.utils.logger import setup\_logging

from src.utils.config import Config

# Создай класс TelegramBot с методами:

# - \_\_init\_\_(self, token: str, config: Config)

# - setup\_handlers(self)

# - run(self)

# - shutdown(self)

# Добавь функцию main() с полной инициализацией

# Добавь обработчик сигналов для graceful shutdown

1. **Обработчик бота (src/bot/handlers.py)**

# Создай полный модуль обработчиков Telegram-бота

# ТРЕБОВАНИЯ:

# 1. Класс BotHandlers с async методами для всех типов сообщений

# 2. Метод start\_handler для команды /start с приветствием

# 3. Метод help\_handler для команды /help с описанием функций

# 4. Метод document\_handler для обработки файлов субтитров

# 5. Метод text\_handler для текстовых сообщений

# 6. Валидация типов файлов (только .srt, .vtt)

# 7. Проверка размера файлов (максимум 20MB)

# 8. Уведомления о статусе обработки

# 9. Обработка ошибок с пользовательскими сообщениями

# 10. Логгирование всех действий пользователей

"""

Обработчики событий Telegram-бота для работы с субтитрами.

Модуль содержит все обработчики команд и сообщений бота.

"""

import logging

from typing import Optional

from telegram import Update, Document

from telegram.ext import ContextTypes

from telegram.constants import ParseMode

from ..services.ai\_service import AIService

from ..services.subtitle\_parser import SubtitleParser

from ..services.template\_manager import TemplateManager

from ..utils.file\_handler import FileHandler

# Создай класс BotHandlers со всеми необходимыми методами

# Каждый метод должен:

# - Принимать update: Update и context: ContextTypes.DEFAULT\_TYPE

# - Включать try-except блоки

# - Логгировать действия

# - Отправлять статусные сообщения пользователю

# - Возвращать корректные статусы

1. **AI сервис (src/services/ai\_service.py)**

# Создай сервис для интеграции с OpenAI API

# ТРЕБОВАНИЯ:

# 1. Класс AIService с инициализацией OpenAI клиента

# 2. Метод analyze\_subtitles(subtitle\_content: str, template: str) -> str

# 3. Настройка модели GPT-4o-mini для оптимизации стоимости

# 4. Система промптов для создания монтажных листов

# 5. Обработка rate limits и ошибок API

# 6. Retry механизм с экспоненциальной задержкой

# 7. Валидация ответов от AI

# 8. Логгирование всех запросов и ответов

# 9. Конфигурируемые параметры (temperature, max\_tokens)

# 10. Кэширование частых запросов

"""

Сервис для работы с OpenAI API и генерации монтажных листов.

Обеспечивает интеграцию с языковыми моделями для анализа субтитров.

"""

import asyncio

import logging

from typing import Optional, Dict, Any

import openai

from openai import AsyncOpenAI

import hashlib

import json

from datetime import datetime, timedelta

# Создай класс AIService с методами:

# - \_\_init\_\_(self, api\_key: str, model: str = "gpt-4o-mini")

# - async analyze\_subtitles(self, content: str, template: str) -> str

# - async \_make\_request\_with\_retry(self, messages: list) -> str

# - \_create\_system\_prompt(self, template: str) -> str

# - \_validate\_response(self, response: str) -> bool

# - \_cache\_key(self, content: str) -> str

# Добавь константы для системных промптов

SYSTEM\_PROMPT\_TEMPLATE = """

Ты профессиональный видеомонтажер и помощник по созданию монтажных листов.

Твоя задача - проанализировать субтитры видео и создать детальный монтажный лист.

ПРАВИЛА:

1. Анализируй диалоги и определяй ключевые сцены

2. Создавай структурированный монтажный лист с тайм-кодами

3. Включай описания действий и переходов

4. Следуй формату предоставленного шаблона

5. Используй профессиональную терминологию

ШАБЛОН ДЛЯ СЛЕДОВАНИЯ:

{template}

ФОРМАТ ОТВЕТА:

- Возвращай только готовый монтажный лист

- Не добавляй комментарии или объяснения

- Соблюдай структуру шаблона

"""

1. **Парсер субтитров (src/services/subtitle\_parser.py)**

# Создай универсальный парсер субтитров для форматов SRT и VTT

# ТРЕБОВАНИЯ:

# 1. Класс SubtitleParser с поддержкой множественных форматов

# 2. Автоматическое определение формата файла

# 3. Парсинг временных меток и текста

# 4. Объединение коротких фраз в логические блоки

# 5. Очистка текста от форматирования

# 6. Обработка различных кодировок (UTF-8, CP1251)

# 7. Валидация структуры субтитров

# 8. Экспорт в удобный для AI формат

# 9. Статистика по субтитрам (длительность, количество блоков)

# 10. Обработка ошибок формата

"""

Парсер субтитров для обработки файлов SRT и VTT форматов.

Обеспечивает универсальную обработку различных форматов субтитров.

"""

import re

import logging

from typing import List, Dict, Optional, Tuple

from dataclasses import dataclass

from datetime import datetime, timedelta

import chardet

import pysrt

from subtitle\_parser import SrtParser, WebVttParser

@dataclass

class SubtitleEntry:

"""Структура данных для одной записи субтитров."""

start\_time: str

end\_time: str

text: str

duration: float

# Создай класс SubtitleParser с методами:

# - \_\_init\_\_(self)

# - parse\_file(self, file\_path: str) -> List[SubtitleEntry]

# - \_detect\_format(self, file\_path: str) -> str

# - \_detect\_encoding(self, file\_path: str) -> str

# - \_parse\_srt(self, content: str) -> List[SubtitleEntry]

# - \_parse\_vtt(self, content: str) -> List[SubtitleEntry]

# - \_clean\_text(self, text: str) -> str

# - \_merge\_short\_entries(self, entries: List[SubtitleEntry]) -> List[SubtitleEntry]

# - get\_statistics(self, entries: List[SubtitleEntry]) -> Dict[str, Any]

# - format\_for\_ai(self, entries: List[SubtitleEntry]) -> str

1. **менеджер шаблонов (src/services/template\_manager.py)**

# Создай менеджер шаблонов монтажных листов

# ТРЕБОВАНИЯ:

# 1. Класс TemplateManager для загрузки и управления шаблонами

# 2. Поддержка множественных шаблонов (по жанрам, типам видео)

# 3. Валидация структуры шаблонов

# 4. Кэширование загруженных шаблонов

# 5. Возможность обновления шаблонов без перезапуска

# 6. Система переменных в шаблонах

# 7. Статистика использования шаблонов

# 8. Резервные шаблоны при ошибках

# 9. Экспорт и импорт шаблонов

# 10. Валидация соответствия AI требованиям

"""

Менеджер шаблонов для создания монтажных листов.

Управляет загрузкой, валидацией и предоставлением шаблонов.

"""

import os

import json

import logging

from typing import Dict, List, Optional, Any

from pathlib import Path

import yaml

from datetime import datetime

# Создай класс TemplateManager с методами:

# - \_\_init\_\_(self, templates\_dir: str)

# - load\_templates(self) -> None

# - get\_template(self, template\_name: str = "default") -> str

# - list\_available\_templates(self) -> List[str]

# - validate\_template(self, template\_content: str) -> bool

# - reload\_templates(self) -> None

# - get\_template\_info(self, template\_name: str) -> Dict[str, Any]

# - \_process\_template\_variables(self, template: str, variables: Dict) -> str

# Добавь константу с базовым шаблоном:

DEFAULT\_TEMPLATE = """

МОНТАЖНЫЙ ЛИСТ

Название: {project\_name}

Дата: {date}

Длительность: {duration}

СТРУКТУРА СЦЕН:

{scenes}

ТЕХНИЧЕСКИЕ ПРИМЕЧАНИЯ:

{technical\_notes}

"""

1. **Утилиты (src/utils/)**

# Создай модуль utils/logger.py для настройки логгирования

# ТРЕБОВАНИЯ:

# 1. Функция setup\_logging() с конфигурацией разных уровней

# 2. Ротация логов по размеру и времени

# 3. Различные форматы для консоли и файла

# 4. Фильтрация чувствительных данных (токены, API ключи)

# 5. Структурированное логгирование в JSON формате

# 6. Интеграция с системой мониторинга

# 7. Кастомные логгеры для разных модулей

# 8. Асинхронное логгирование для производительности

"""

Конфигурация системы логгирования приложения.

"""

import logging

import logging.handlers

import sys

import json

from datetime import datetime

from pathlib import Path

# Создай функцию setup\_logging(log\_level: str = "INFO", log\_file: str = "bot.log")

# Создай модуль utils/config.py для управления конфигурацией

# ТРЕБОВАНИЯ:

# 1. Класс Config с загрузкой из .env и config.yaml

# 2. Валидация обязательных параметров

# 3. Типизированные настройки

# 4. Значения по умолчанию

# 5. Возможность переопределения через переменные окружения

# 6. Секретные параметры (токены, ключи)

# 7. Настройки для разных окружений (dev, prod)

"""

Управление конфигурацией приложения.

"""

import os

from typing import Optional, Dict, Any

from dataclasses import dataclass

import yaml

from dotenv import load\_dotenv

@dataclass

class Config:

telegram\_token: str

openai\_api\_key: str

log\_level: str

max\_file\_size: int

templates\_dir: str

cache\_ttl: int

rate\_limit\_requests: int

rate\_limit\_window: int

# Создай функцию load\_config() -> Config

1. **Тесты**

# Создай comprehensive test suite в директории tests/

# ТРЕБОВАНИЯ:

# 1. Тесты для всех основных модулей

# 2. Mock объекты для внешних API

# 3. Тестирование асинхронных функций

# 4. Fixtures для тестовых данных

# 5. Тесты интеграции с Telegram API

# 6. Тесты обработки различных форматов файлов

# 7. Тесты обработки ошибок

# 8. Performance тесты

# 9. Покрытие кода не менее 80%

# Создай файл tests/test\_bot.py

"""

Тесты для основных функций Telegram-бота.

"""

import pytest

import asyncio

from unittest.mock import AsyncMock, MagicMock, patch

from telegram import Update, Message, Document, User, Chat

from telegram.ext import ContextTypes

from src.bot.handlers import BotHandlers

from src.services.ai\_service import AIService

# Создай pytest fixtures и тестовые классы

# Используй async/await для асинхронных тестов

# Добавь параметризованные тесты для различных сценариев

1. README.md

# Создай подробный README.md файл

# ТРЕБОВАНИЯ:

# 1. Описание проекта и его возможностей

# 2. Инструкции по установке и настройке

# 3. Примеры использования с скриншотами

# 4. Документация API

# 5. Руководство по развертыванию

# 6. Troubleshooting секция

# 7. Информация для разработчиков

# 8. License и контакты

# Telegram Subtitle Processing Bot

Автоматизированный Telegram-бот для обработки субтитров и генерации монтажных листов с использованием искусственного интеллекта.

## Возможности

- ✅ Обработка файлов субтитров (SRT, VTT)

- 🤖 Генерация монтажных листов через OpenAI GPT-4

- 📊 Анализ содержимого видео по субтитрам

- 🎬 Профессиональные шаблоны монтажных листов

- ⚡ Асинхронная обработка запросов

- 🔒 Безопасная обработка файлов

- 📝 Подробное логгирование

- 🧪 Полное покрытие тестами

## Быстрый старт

### Установка

# Добавь детальные инструкции по установке, настройке, использованию