Система автопостинга в Telegram с RAG и LM Studio

Общее описание

Разработать Python-приложение для автоматического создания и публикации контента в Telegram-канал с использованием RAG-системы (Retrieval-Augmented Generation) и локальной языковой модели через LM Studio.

Архитектура системы

Структура проекта

```
project_root/
├── main.py                        # Точка входа, управление жизненным циклом
├── logs.py                        # Централизованное логирование
├── config/
│   ├── config.json                # Основная конфигурация
│   ├── telegram_token.txt         # Токен Telegram-бота
│   └── telegram_channel.txt       # ID канала
├── data/
│   ├── prompt_1/                  # Шаблоны первой части промпта
│   ├── prompt_2/                  # Шаблоны второй части промпта
│   ├── prompt_3/                  # Шаблоны третьей части промпта
│   ├── topics.txt                # Список тем для обработки
│   └── state.json                # Состояние системы
├── inform/                        # База знаний для RAG
├── media/                         # Медиафайлы для публикации
├── modules/
│   ├── rag_system/
│   │   ├── rag_retriever.py      # Гибридный поиск и индексация
│   │   ├── rag_file_utils.py     # Извлечение текста из файлов
│   │   ├── rag_chunk_tracker.py # Трекинг использования чанков
│   │   └── embedding_manager.py  # Управление embedding-моделью
│   ├── content_generation/
│   │   ├── prompt_builder.py     # Сборка промптов
│   │   ├── lm_client.py          # Клиент для LM Studio
│   │   └── content_validator.py # Валидация контента
│   ├── external_apis/
│   │   ├── web_search.py         # Интеграция с serper.dev
│   │   └── telegram_client.py    # Публикация в Telegram
│   └── utils/
│       ├── config_manager.py     # Управление конфигурацией
│       ├── file_processor.py     # Обработка файлов разных форматов
│       └── media_handler.py      # Работа с медиафайлами
└── requirements.txt
```

Детальное техническое задание

1. Инициализация и конфигурация

1.1 Создание config_manager.py

python

```python
class ConfigManager:
    def load_config(self) -> dict
    def get_telegram_token(self) -> str
    def get_telegram_channel_id(self) -> str
    def get_lm_studio_config(self) -> dict
    def get_rag_config(self) -> dict
    def get_serper_api_key(self) -> str
```

1.2 Структура config.json

json

```json
{
    "lm_studio": {
        "base_url": "http://localhost:1234/v1",
        "model": "qwen2.5-14b",
        "max_tokens": 4096,
        "temperature": 0.7,
        "timeout": 1200
    },
    "rag": {
        "embedding_model": "all-MiniLM-L6-v2",
        "chunk_size": 512,
        "chunk_overlap": 50,
        "max_context_length": 4096,
        "media_context_length": 1024,
        "similarity_threshold": 0.7
    },
    "telegram": {
        "post_interval": 900,
        "max_retries": 3
    },
    "serper": {
        "results_limit": 10
    },
    "processing": {
        "batch_size": 10,
        "max_file_size_mb": 50
    }
}
```

2. RAG-система

2.1 Обработка файлов (file_processor.py)

python

```python
class FileProcessor:
    def extract_text_from_file(self, file_path: str) -> str
    def process_csv(self, file_path: str) -> str
    def process_xlsx(self, file_path: str) -> str
    def process_pdf(self, file_path: str) -> str
    def process_docx(self, file_path: str) -> str
    def process_html(self, file_path: str) -> str
    def process_txt(self, file_path: str) -> str
    def clean_text(self, text: str) -> str
    def normalize_encoding(self, text: str) -> str
```

Требования к обработке:

- Удаление HTML-тегов
- Очистка от пустых ячеек (для таблиц)
- Нормализация кодировки в UTF-8
- Фильтрация мусорных данных
- Сохранение структуры таблиц в читаемом формате

2.2 Embedding и индексация (embedding_manager.py)

python

```python
class EmbeddingManager:
    def __init__(self, model_name: str = "all-MiniLM-L6-v2")
    def encode_texts(self, texts: list) -> np.ndarray
    def build_faiss_index(self, embeddings: np.ndarray) -> faiss.Index
    def save_index(self, index: faiss.Index, path: str)
    def load_index(self, path: str) -> faiss.Index
    def search_similar(self, query: str, k: int = 5) -> list
```

2.3 Чанкинг и индексация (rag_retriever.py)

python

```python
class RAGRetriever:
    def __init__(self, config: dict)
    def process_inform_folder(self, folder_path: str)
    def chunk_text(self, text: str, chunk_size: int = 512) -> list
    def build_knowledge_base(self)
    def retrieve_context(self, query: str, max_length: int = 4096) -> str
    def update_knowledge_base(self, new_content: str)
    def get_relevant_chunks(self, topic: str, limit: int = 10) -> list
```

Алгоритм работы:

1. Сканирование папки `inform/`
2. Извлечение текста из всех поддерживаемых форматов
3. Разбиение на чанки с перекрытием
4. Создание эмбеддингов для каждого чанка
5. Построение FAISS индекса
6. Сохранение индекса в `faiss_index.idx`

2.4 Трекинг использования (rag_chunk_tracker.py)

```python
class ChunkTracker:
    def __init__(self)
    def track_usage(self, chunk_id: str, topic: str)
    def get_usage_penalty(self, chunk_id: str) -> float
    def reset_usage_stats(self)
    def get_diverse_chunks(self, candidates: list) -> list
```

3. Интеграция с внешними API

3.1 Web-поиск (web_search.py)

```python
class WebSearchClient:
    def __init__(self, api_key: str)
    def search(self, query: str, num_results: int = 10) -> list
    def extract_content(self, search_results: list) -> str
    def save_to_inform(self, content: str, topic: str)
    def format_search_context(self, results: list) -> str
```

Интеграция с serper.dev:

1. Формирование поискового запроса на основе темы
2. Получение результатов через API
3. Извлечение релевантного контента
4. Сохранение в папку `inform/` для пополнения базы знаний
5. Форматирование для включения в контекст

3.2 LM Studio клиент (lm_client.py)

python

```python
class LMStudioClient:
    def __init__(self, base_url: str, model: str)
    def generate_content(self, prompt: str, max_tokens: int = 4096) -> str
    def check_connection(self) -> bool
    def retry_generation(self, prompt: str, max_retries: int = 3) -> str
    def validate_response_length(self, text: str, max_length: int) -> bool
```

3.3 Telegram клиент (telegram_client.py)

python

```python
class TelegramClient:
    def __init__(self, token: str, channel_id: str)
    def send_text_message(self, text: str) -> bool
    def send_media_message(self, text: str, media_path: str) -> bool
    def validate_message_length(self, text: str, has_media: bool) -> bool
    def format_message(self, text: str) -> str
```

4. Генерация контента

4.1 Сборка промптов (prompt_builder.py)

python

```python
class PromptBuilder:
    def __init__(self, prompt_folders: list)
    def load_prompt_templates(self)
    def select_random_templates(self) -> tuple
    def build_prompt(self, topic: str, context: str, media_file: str = None) -> str
    def replace_placeholders(self, template: str, replacements: dict) -> str
    def validate_prompt_structure(self, prompt: str) -> bool
```

Алгоритм сборки промпта:

1. Выбор случайного файла из каждой папки (`prompt_1/`, `prompt_2/`, `prompt_3/`)
2. Объединение содержимого файлов
3. Замена плейсхолдеров:
   - `{TOPIC}` → текущая тема из `topics.txt`
   - `{CONTEXT}` → релевантный контекст из RAG + web-поиск
   - `{UPLOADFILE}` → путь к медиафайлу (если есть)
4. Проверка корректности структуры

4.2 Валидация контента (content_validator.py)

python

```python
class ContentValidator:
    def __init__(self, config: dict)
    def validate_length(self, text: str, has_media: bool) -> bool
    def remove_tables(self, text: str) -> str
    def remove_thinking_blocks(self, text: str) -> str
    def clean_html_markdown(self, text: str) -> str
    def request_shorter_version(self, original_prompt: str) -> str
    def validate_content_quality(self, text: str) -> bool
```

Правила валидации:

- Максимум 4096 символов для текста без медиа
- Максимум 1024 символа для подписи к медиа
- Удаление таблиц (Markdown и HTML)
- Удаление блоков размышлений `<think>...</think>`
- Запрос на сокращение при превышении лимитов

4.3 Обработка медиа (media_handler.py)

python

```python
class MediaHandler:
    def __init__(self, media_folder: str)
    def get_random_media_file(self) -> str
    def validate_media_file(self, file_path: str) -> bool
    def get_media_type(self, file_path: str) -> str
    def resize_image_if_needed(self, file_path: str) -> str
    def get_supported_formats(self) -> list
```

5. Главный цикл обработки (main.py)

5.1 Инициализация системы

python

```python
class TelegramRAGSystem:
    def __init__(self, config_path: str)
    def initialize_components(self)
    def setup_logging(self)
    def validate_configuration(self)
    def build_initial_knowledge_base(self)
```

5.2 Основной цикл

python

```python
def main_processing_loop(self):
    while True:
        topic = self.get_next_topic()
        if not topic:
            break

        try:
            # 1. Поиск контекста в RAG
            rag_context = self.rag_retriever.retrieve_context(topic)

            # 2. Web-поиск дополнительной информации
            web_context = self.web_search.search_and_extract(topic)

            # 3. Объединение контекста
            full_context = self.combine_contexts(rag_context, web_context)

            # 4. Выбор медиафайла (опционально)
            media_file = self.media_handler.get_random_media_file()

            # 5. Сборка промпта
            prompt = self.prompt_builder.build_prompt(
                topic=topic,
                context=full_context,
                media_file=media_file
            )

            # 6. Генерация контента
            content = self.lm_client.generate_content(prompt)

            # 7. Валидация и очистка
            validated_content = self.content_validator.process(content)

            # 8. Публикация в Telegram
            success = self.telegram_client.publish(validated_content, media_file)

            # 9. Обновление состояния
            self.update_processing_state(topic, success)

            # 10. Пауза перед следующей итерацией
            time.sleep(self.config['telegram']['post_interval'])

        except Exception as e:
            self.handle_error(topic, e)
            continue
```

6. Обработка ошибок и мониторинг

6.1 Система логирования (logs.py)

python

```python
def setup_logger(name: str, log_file: str, level=logging.INFO) -> logging.Logger
def log_processing_stats(topics_processed: int, errors: int, success_rate: float)
def log_rag_performance(retrieval_time: float, context_length: int)
def log_telegram_status(message_sent: bool, error_details: str = None)
```

6.2 Состояние системы (state_manager.py)

python

```python
class StateManager:
    def __init__(self, state_file: str)
    def load_state(self) -> dict
    def save_state(self, state: dict)
    def mark_topic_processed(self, topic: str, success: bool)
    def get_unprocessed_topics(self) -> list
    def get_processing_statistics(self) -> dict
    def reset_failed_topics(self)
```

7. Требования к зависимостям (requirements.txt)

```txt
# Основные библиотеки
sentence-transformers==2.2.2
faiss-cpu==1.7.4
openai==1.12.0
requests==2.31.0
python-telegram-bot==20.7

# Обработка файлов
pandas==2.1.4
openpyxl==3.1.2
PyPDF2==3.0.1
python-docx==1.1.0
beautifulsoup4==4.12.2
lxml==4.9.3

# Утилиты
numpy==1.26.2
tqdm==4.66.1
python-dotenv==1.0.0
schedule==1.2.0
```

8. Дополнительные требования

8.1 Производительность

- Кэширование эмбеддингов
- Батчевая обработка файлов
- Оптимизация FAISS индекса
- Ограничение использования памяти

8.2 Надежность

- Graceful shutdown при получении сигналов
- Восстановление после сбоев
- Валидация всех входных данных
- Резервное копирование состояния

8.3 Мониторинг

- Детальное логирование всех операций
- Метрики производительности
- Статистика использования API

- Отчеты об ошибках

8.4 Безопасность

- Безопасное хранение API-ключей
- Валидация путей к файлам
- Ограничение размеров файлов
- Санитизация входных данных

9. Тестирование

9.1 Модульные тесты

- Тестирование каждого компонента изолированно
- Мокирование внешних API
- Проверка обработки ошибок
- Валидация форматов данных

9.2 Интеграционные тесты

- Тестирование полного цикла обработки
- Проверка взаимодействия с LM Studio
- Валидация публикации в Telegram
- Тестирование RAG-системы

10. Развертывание и запуск

10.1 Подготовка окружения

1. Установка Python 3.13
2. Создание виртуального окружения
3. Установка зависимостей
4. Настройка LM Studio
5. Конфигурирование API-ключей

10.2 Первоначальная настройка

1. Подготовка папки `inform/` с базовыми данными
2. Создание шаблонов промптов
3. Настройка списка тем
4. Тестирование подключений к API
5. Запуск инициализации RAG-системы

Эта архитектура обеспечивает модульность, масштабируемость и надежность системы автопостинга с полноценной RAG-интеграцией.