Карта файлов и функций системы автопостинга

Основные файлы системы

main.py - Точка входа и управление системой

Назначение: Главный файл для запуска и координации всей системы

Классы:

- `TelegramRAGSystem` - Главный класс системы

Функции:

```python
def __init__(self, config_path: str)                          # Инициализация системы
def initialize_components(self)                               # Инициализация всех компонентов
def setup_logging(self)                                       # Настройка логирования
def validate_configuration(self)                              # Проверка конфигурации
def build_initial_knowledge_base(self)                        # Построение базы знаний
def get_next_topic(self) -> str                               # Получение следующей темы
def combine_contexts(self, rag_context: str, web_context: str) -> str  # Объединение конте
def main_processing_loop(self)                                # Основной цикл обработки
def handle_error(self, topic: str, error: Exception)          # Обработка ошибок
def update_processing_state(self, topic: str, success: bool)  # Обновление состояния
def graceful_shutdown(self)                                   # Корректное завершение
def run(self)                                                 # Запуск системы
```

---

logs.py - Система логирования

Назначение: Централизованная настройка и управление логами

Функции:

```python
def setup_logger(name: str, log_file: str, level=logging.INFO) -> logging.Logger  # Создан
def setup_file_handler(log_file: str, level=logging.INFO) -> logging.FileHandler  # Файлов
def setup_console_handler(level=logging.INFO) -> logging.StreamHandler            # Консол
def get_logger(name: str) -> logging.Logger                                       # Получе
def log_processing_stats(topics_processed: int, errors: int, success_rate: float)  # Стати
def log_rag_performance(retrieval_time: float, context_length: int)               # Произв
def log_telegram_status(message_sent: bool, error_details: str = None)            # Статус
def log_system_info(system_info: dict)                                            # Системн
```

Модули RAG-системы

modules/rag_system/rag_retriever.py - Поиск и индексация

Назначение: Гибридный поиск по базе знаний с FAISS индексом

Классы:

- `RAGRetriever` - Основной класс для работы с поиском

Функции:

```python
def __init__(self, config: dict)                          # Инициализация
def process_inform_folder(self, folder_path: str)         # Обработка папки с данными
def chunk_text(self, text: str, chunk_size: int = 512) -> list  # Разбиение на чанки
def build_knowledge_base(self)                            # Построение базы знаний
def create_embeddings(self, texts: list) -> np.ndarray # Создание эмбеддингов
def build_faiss_index(self, embeddings: np.ndarray)   # Построение FAISS индекса
def save_index(self, index_path: str)                     # Сохранение индекса
def load_index(self, index_path: str) -> bool             # Загрузка индекса
def retrieve_context(self, query: str, max_length: int = 4096) -> str  # Поиск контекста
def get_relevant_chunks(self, topic: str, limit: int = 10) -> list      # Получение релева
def update_knowledge_base(self, new_content: str, source: str)          # Обновление базы
def search_similar(self, query: str, k: int = 5) -> list                # Семантический пои
def rerank_results(self, query: str, candidates: list) -> list          # Переранжирование
def get_index_stats(self) -> dict                         # Статистика индекса
```

modules/rag_system/rag_file_utils.py - Обработка файлов

Назначение: Извлечение текста из файлов разных форматов

Классы:

- `FileProcessor` - Процессор файлов разных форматов

Функции:

```python
def __init__(self)                                        # Инициализация
def extract_text_from_file(self, file_path: str) -> str  # Универсальное извлечение тексто
def process_txt(self, file_path: str) -> str              # Обработка TXT файлов
def process_csv(self, file_path: str) -> str              # Обработка CSV файлов
def process_xlsx(self, file_path: str) -> str             # Обработка Excel файлов
def process_pdf(self, file_path: str) -> str              # Обработка PDF файлов
def process_docx(self, file_path: str) -> str             # Обработка Word файлов
def process_html(self, file_path: str) -> str             # Обработка HTML файлов
def detect_encoding(self, file_path: str) -> str          # Определение кодировки
def clean_text(self, text: str) -> str                    # Очистка текста
def normalize_encoding(self, text: str) -> str            # Нормализация кодировки
def remove_html_tags(self, text: str) -> str              # Удаление HTML тегов
def filter_empty_cells(self, data: list) -> list          # Фильтрация пустых ячеек
def get_supported_formats(self) -> list                   # Поддерживаемые форматы
def validate_file(self, file_path: str) -> bool           # Валидация файла
```

modules/rag_system/rag_chunk_tracker.py - Трекинг чанков

Назначение: Отслеживание использования чанков знаний для разнообразия

Классы:

- `ChunkTracker` - Трекер использования чанков

Функции:

```python
def __init__(self)                                    # Инициализация
def track_usage(self, chunk_id: str, topic: str)      # Отслеживание использования
def get_usage_penalty(self, chunk_id: str) -> float   # Получение штрафа за использование
def get_usage_count(self, chunk_id: str) -> int       # Количество использований
def reset_usage_stats(self)                           # Сброс статистики
def get_diverse_chunks(self, candidates: list) -> list  # Получение разнообразных чанков
def apply_penalty_scores(self, chunks: list) -> list   # Применение штрафных очков
def get_tracker_stats(self) -> dict                   # Статистика трекера
def save_usage_data(self, file_path: str)             # Сохранение данных об использовани
def load_usage_data(self, file_path: str)             # Загрузка данных об использовании
def cleanup_old_usage(self, days_threshold: int = 30) # Очистка старых данных
```

modules/rag_system/embedding_manager.py - Управление эмбеддингами

Назначение: Работа с embedding моделью all-MiniLM-L6-v2

Классы:

- `EmbeddingManager` - Менеджер эмбеддингов

Функции:

```python
def __init__(self, model_name: str = "all-MiniLM-L6-v2")  # Инициализация
def load_model(self)                                      # Загрузка модели
def encode_texts(self, texts: list) -> np.ndarray      # Кодирование текстов
def encode_single_text(self, text: str) -> np.ndarray  # Кодирование одного текста
def batch_encode(self, texts: list, batch_size: int = 32) -> np.ndarray  # Батчевое кодир
def calculate_similarity(self, emb1: np.ndarray, emb2: np.ndarray) -> float  # Расчет схо
def find_most_similar(self, query_emb: np.ndarray, candidate_embs: np.ndarray, k: int) ->
def get_model_info(self) -> dict                       # Информация о модели
def save_embeddings(self, embeddings: np.ndarray, file_path: str)  # Сохранение эмбеддинг
def load_embeddings(self, file_path: str) -> np.ndarray  # Загрузка эмбеддингов
```

Модули генерации контента

modules/content_generation/prompt_builder.py - Сборка промптов

Назначение: Создание промптов из шаблонов с заменой плейсхолдеров

Классы:

- `PromptBuilder` - Конструктор промптов

Функции:

```python
def __init__(self, prompt_folders: list)              # Инициализация
def load_prompt_templates(self)                       # Загрузка шаблонов
def scan_prompt_folder(self, folder_path: str) -> list # Сканирование папки с промптами
def select_random_templates(self) -> tuple            # Выбор случайных шаблонов
def read_template_file(self, file_path: str) -> str   # Чтение файла шаблона
def build_prompt(self, topic: str, context: str, media_file: str = None) -> str  # Сборка
def replace_placeholders(self, template: str, replacements: dict) -> str  # Замена плейсхо
def validate_prompt_structure(self, prompt: str) -> bool  # Валидация структуры промпта
def check_placeholder_presence(self, template: str) -> dict  # Проверка наличия плейсхолд
def get_template_stats(self) -> dict                  # Статистика шаблонов
def reload_templates(self)                            # Перезагрузка шаблонов
def test_template_combination(self, topic: str, context: str) -> str  # Тестирование комби
```

modules/content_generation/lm_client.py - Клиент LM Studio

Назначение: Взаимодействие с языковой моделью через LM Studio API

Классы:

- `LMStudioClient` - Клиент для LM Studio

Функции:

```python
def __init__(self, base_url: str, model: str, config: dict)  # Инициализация
def check_connection(self) -> bool                    # Проверка подключения
def get_model_info(self) -> dict                      # Информация о модели
def generate_content(self, prompt: str, max_tokens: int = 4096) -> str  # Генерация конте
def generate_with_retry(self, prompt: str, max_retries: int = 3) -> str  # Генерация с по
def request_shorter_version(self, original_prompt: str, current_length: int, target_lengt
def validate_response_length(self, text: str, max_length: int) -> bool  # Валидация длины
def estimate_tokens(self, text: str) -> int           # Оценка количества токенов
def set_generation_parameters(self, temperature: float, top_p: float, top_k: int)  # Наст
def get_generation_stats(self) -> dict                # Статистика генерации
def clear_conversation_history(self)                  # Очистка истории диалога
def health_check(self) -> dict                        # Проверка здоровья системы
```

`modules/content_generation/content_validator.py` - Валидация контента

Назначение: Проверка и очистка сгенерированного контента

Классы:

- `ContentValidator` - Валидатор контента

Функции:

```python
def __init__(self, config: dict)                     # Инициализация
def validate_content(self, text: str, has_media: bool = False) -> str  # Полная валидация
def validate_length(self, text: str, has_media: bool) -> bool  # Проверка длины
def remove_tables(self, text: str) -> str             # Удаление таблиц
def remove_thinking_blocks(self, text: str) -> str    # Удаление блоков размышлений
def clean_html_markdown(self, text: str) -> str       # Очистка HTML/Markdown
def remove_markdown_tables(self, text: str) -> str    # Удаление таблиц Markdown
def remove_html_tables(self, text: str) -> str        # Удаление HTML таблиц
def detect_thinking_patterns(self, text: str) -> list  # Обнаружение паттернов размышлени
def clean_special_characters(self, text: str) -> str  # Очистка спецсимволов
def validate_content_quality(self, text: str) -> bool  # Проверка качества контента
def get_content_stats(self, text: str) -> dict        # Статистика контента
def format_for_telegram(self, text: str) -> str       # Форматирование для Telegram
```

Модули внешних API

`modules/external_apis/web_search.py` - Web-поиск

Назначение: Поиск информации в интернете через serper.dev API

Классы:

- `WebSearchClient` - Клиент для web-поиска

Функции:

```python
def __init__(self, api_key: str, config: dict)          # Инициализация
def search(self, query: str, num_results: int = 10) -> list  # Поиск по запросу
def build_search_query(self, topic: str) -> str         # Построение поискового запроса
def extract_content(self, search_results: list) -> str # Извлечение контента из результат
def filter_relevant_results(self, results: list, topic: str) -> list  # Фильтрация релева
def save_to_inform(self, content: str, topic: str, source: str)  # Сохранение в базу знан
def format_search_context(self, results: list) -> str  # Форматирование контекста
def get_search_stats(self) -> dict                      # Статистика поиска
def validate_search_results(self, results: list) -> bool  # Валидация результатов
def handle_rate_limits(self, response: dict) -> bool    # Обработка лимитов API
def clean_search_content(self, content: str) -> str     # Очистка контента
def deduplicate_results(self, results: list) -> list    # Удаление дубликатов
```

modules/external_apis/telegram_client.py - Telegram API

Назначение: Публикация сообщений и медиа в Telegram канал

Классы:

- `TelegramClient` - Клиент для Telegram

Функции:

```python
def __init__(self, token: str, channel_id: str, config: dict)  # Инициализация
def send_text_message(self, text: str) -> bool          # Отправка текстового сообщения
def send_media_message(self, text: str, media_path: str) -> bool  # Отправка медиа сообще
def send_photo(self, photo_path: str, caption: str) -> bool  # Отправка фото
def send_video(self, video_path: str, caption: str) -> bool  # Отправка видео
def send_document(self, doc_path: str, caption: str) -> bool  # Отправка документа
def validate_message_length(self, text: str, has_media: bool) -> bool  # Валидация длины
def format_message(self, text: str) -> str              # Форматирование сообщения
def escape_markdown(self, text: str) -> str             # Экранирование Markdown
def check_bot_permissions(self) -> dict                 # Проверка прав бота
def get_channel_info(self) -> dict                      # Информация о канале
def handle_telegram_errors(self, error: Exception) -> bool  # Обработка ошибок Telegram
def retry_send_message(self, message_data: dict, max_retries: int = 3) -> bool  # Повтор
def get_send_stats(self) -> dict                        # Статистика отправки
```

Утилиты

modules/utils/config_manager.py - Управление конфигурацией

Назначение: Загрузка и управление настройками системы

Классы:

- `ConfigManager` - Менеджер конфигурации

Функции:

```python
def __init__(self, config_path: str)                    # Инициализация
def load_config(self) -> dict                           # Загрузка конфигурации
def get_telegram_token(self) -> str                     # Получение токена Telegram
def get_telegram_channel_id(self) -> str                # Получение ID канала
def get_lm_studio_config(self) -> dict                  # Конфигурация LM Studio
def get_rag_config(self) -> dict                        # Конфигурация RAG
def get_serper_api_key(self) -> str                     # Ключ API serper.dev
def validate_config(self) -> bool                       # Валидация конфигурации
def get_config_value(self, key_path: str, default=None)  # Получение значения по пути
def update_config_value(self, key_path: str, value)     # Обновление значения
def save_config(self)                                   # Сохранение конфигурации
def reload_config(self)                                 # Перезагрузка конфигурации
def get_all_config(self) -> dict                        # Получение всей конфигурации
```

modules/utils/file_processor.py - Обработчик файлов

Назначение: Универсальная обработка файлов различных форматов

Классы:

- `UniversalFileProcessor` - Универсальный процессор файлов

Функции:

```python
def __init__(self, config: dict)                        # Инициализация
def process_file(self, file_path: str) -> dict          # Обработка файла
def get_file_type(self, file_path: str) -> str          # Определение типа файла
def validate_file_size(self, file_path: str) -> bool    # Проверка размера файла
def extract_metadata(self, file_path: str) -> dict      # Извлечение метаданных
def process_batch(self, file_paths: list) -> list       # Батчевая обработка
def get_processing_stats(self) -> dict                  # Статистика обработки
def cleanup_temp_files(self)                           # Очистка временных файлов
def handle_processing_error(self, file_path: str, error: Exception)  # Обработка ошибок
def get_supported_extensions(self) -> list              # Поддерживаемые расширения
def estimate_processing_time(self, file_paths: list) -> int  # Оценка времени обработки
```

modules/utils/media_handler.py - Обработчик медиа

Назначение: Работа с медиафайлами для публикации

Классы:

- `MediaHandler` - Обработчик медиафайлов

Функции:

```python
def __init__(self, media_folder: str, config: dict)    # Инициализация
def get_random_media_file(self) -> str                  # Получение случайного медиафайла
def get_media_files_list(self) -> list                  # Список медиафайлов
def validate_media_file(self, file_path: str) -> bool  # Валидация медиафайла
def get_media_type(self, file_path: str) -> str         # Определение типа медиа
def get_file_size(self, file_path: str) -> int          # Размер файла
def resize_image_if_needed(self, file_path: str, max_size: tuple) -> str  # Изменение раз
def compress_video_if_needed(self, file_path: str) -> str  # Сжатие видео
def get_image_dimensions(self, file_path: str) -> tuple # Размеры изображения
def get_video_duration(self, file_path: str) -> float  # Длительность видео
def get_supported_formats(self) -> dict                 # Поддерживаемые форматы
def create_thumbnail(self, file_path: str) -> str       # Создание миниатюры
def get_media_stats(self) -> dict                       # Статистика медиафайлов
def cleanup_processed_media(self)                       # Очистка обработанных медиа
```

modules/utils/state_manager.py - Управление состоянием

Назначение: Отслеживание состояния обработки тем и системы

Классы:

- `StateManager` - Менеджер состояния

Функции:

```python
def __init__(self, state_file: str)                    # Инициализация
def load_state(self) -> dict                           # Загрузка состояния
def save_state(self)                                   # Сохранение состояния
def mark_topic_processed(self, topic: str, success: bool, details: dict = None)  # Отметк
def get_next_unprocessed_topic(self) -> str            # Следующая необработанная тема
def get_unprocessed_topics(self) -> list               # Список необработанных тем
def get_processed_topics(self) -> list                 # Список обработанных тем
def get_failed_topics(self) -> list                    # Список неудачных тем
def reset_failed_topics(self)                          # Сброс неудачных тем
def get_processing_statistics(self) -> dict            # Статистика обработки
def add_processing_stats(self, stats: dict)            # Добавление статистики
def get_system_uptime(self) -> float                   # Время работы системы
def set_system_status(self, status: str)               # Установка статуса системы
def get_last_activity(self) -> datetime                # Последняя активность
def backup_state(self) -> str                          # Резервное копирование состояния
def restore_state(self, backup_path: str) -> bool      # Восстановление состояния
```

Конфигурационные файлы

config/config.json - Основная конфигурация

Назначение: Центральный файл со всеми настройками системы

config/telegram_token.txt - Токен Telegram

Назначение: Токен бота для доступа к Telegram API

config/telegram_channel.txt - ID канала

Назначение: Идентификатор канала для публикации

Файлы данных

data/topics.txt - Список тем

Назначение: Список тем для обработки, по одной теме на строку

data/state.json - Состояние системы

Назначение: Сохранение состояния обработки тем и статистики

data/prompt_1/, prompt_2/, prompt_3/ - Шаблоны промптов

Назначение: Папки с текстовыми файлами-шаблонами для сборки промптов

Папки с данными

inform/ - База знаний

Назначение: Файлы различных форматов для построения RAG-системы

media/ - Медиафайлы

Назначение: Изображения, видео и документы для публикации в постах

faiss_index.idx - Индекс FAISS

Назначение: Сохраненный индекс для быстрого семантического поиска

Общая схема взаимодействия

1. **main.py** координирует работу всех модулей
2. **RAG-система** обрабатывает знания и предоставляет контекст
3. **Генераторы контента** создают и валидируют посты
4. **Внешние API** интегрируются с сервисами
5. **Утилиты** обеспечивают вспомогательную функциональность
6. **Файлы конфигурации** управляют настройками
7. **Данные** хранят состояние и контент системы