Общество с ограниченной ответственностью «УМАРТА»

Сервис для автоматического перевода документов с использованием LLM-моделей «УМАРТА.Переводчик»

Описание функциональных требований к ПО

Листов 49

Москва, 2025 г.

АННОТАЦИЯ

Настоящий документ представляет собой описание функциональных требований к программному обеспечению системы «Сервис для автоматического перевода документов с использованием LLM-моделей "УМАРТА.Переводчик"» (далее – Система).

Документ охватывает аспекты архитектуры и реализации Системы, включая:

- Архитектуру системы, описание её компонентов и используемый технологический стек.
- Детализированное устройство backend-части, включая модули перевода, работу с LLM-провайдерами, аутентификацию, подписки и платежи.
- Архитектуру frontend-интерфейса, управление состоянием и поддержку интернационализации.
- Инфраструктурные решения, такие как контейнеризация, настройка NGINX и хранение данных.
- Механизмы обеспечения надёжности, отказоустойчивости, мониторинга и логирования.
- Подходы к обеспечению безопасности, валидации данных и ограничениям.
- Меры по обеспечению производительности и масштабируемости системы.
- Интеграции с внешними API, включая LLM-провайдеров, платёжные шлюзы и систему уведомлений.
- Метрики, логи и аналитические средства.
- Возможности по развитию системы: поддержка новых форматов, моделей и языков.
- Особенности работы очереди переводов, включая механизмы семафоров, мониторинг и масштабирование под высокие нагрузки.

Документ предназначен для разработчиков, архитекторов и специалистов, участвующих в проектировании, разработке и сопровождении системы, а также может служить технической основой для принятия решений по её развитию и интеграции.

СОДЕРЖАНИЕ

1 ОБЩЕЕ ОПИСАНИЕ И НАЗНАЧЕНИЕ ПРОЕКТА

1.1 НАЗНАЧЕНИЕ

Проект представляет собой сервис для автоматического перевода документов с использованием локальных или удаленных языковых моделей (LLM). Он позволяет зарегистрированным пользователям загружать файлы, переводить их на выбранный язык и скачивать результат.

Система поддерживает:

– Аутентификацию и авторизацию пользователей.
– Управление файлами (загрузка, скачивание, удаление, просмотр истории).
– Поддержку нескольких LLM-провайдеров для перевода.
– Систему подписок и тарифных планов для управления лимитами.
– Панель администратора для управления пользователями, тарифами и настройками системы.

1.2 ТЕХНОЛОГИЧЕСКИЙ СТЕК

– **Backend:** Python 3.11, FastAPI, SQLAlchemy, Pydantic, aiohttp.
– **Frontend:** TypeScript, React, Vite, Ant Design, i18next.
– **База данных:** PostgreSQL 16.
– **Инфраструктура:** Docker, Docker Compose, Nginx.

2 АРХИТЕКТУРА BACKEND

Backend написан на Python с использованием фреймворка FastAPI. Архитектура следует принципам модульности, разделяя логику на независимые компоненты.

2.1 ТОЧКА ВХОДА И ЗАПУСК (`main.py`)

- **Приложение FastAPI:** Создается экземпляр `FastAPI` с настройками для заголовка, описания, документации (Swagger, ReDoc) и кастомными параметрами для UI.
- **Lifespan Manager:** Используется `asynccontextmanager` для управления жизненным циклом приложения. При старте запускается `AsyncIOScheduler` для фоновых задач, при остановке — корректно завершает его работу.
- **Middleware (Промежуточное ПО):**
  - `CORSMiddleware`: Управляет политикой CORS, разрешая запросы с определенных доменов.
  - `TrustedHostMiddleware`: Обеспечивает безопасность, разрешая запросы только с доверенных хостов.
  - `SessionMiddleware`: Включает поддержку сессий.
  - `LoggingMiddleware`: Кастомный middleware для логирования всех входящих запросов и их ответов.
- **Обработчики исключений:** Реализованы глобальные обработчики для кастомных исключений (`PromoCodeException`), стандартных `HTTPException` и всех остальных непредвиденных ошибок (`Exception`) для возврата корректных HTTP-ответов.
- **Маршрутизация:** Все роутеры из модулей (`auth`, `files`, `subscriptions`, `payments`, `system`, `translation`) подключаются к основному приложению с соответствующими префиксами.
- **Инициализация БД:** Перед запуском приложения выполняется функция `wait_for_db`, которая ожидает доступности PostgreSQL, после чего `init_db` создает все необходимые таблицы.
- **Планировщик задач (`apscheduler`):**
  - `process_recurring_subscriptions`: Ежедневно в 03:00 UTC проверяет и обрабатывает рекуррентные подписки.
  - `cleanup_expired_temp_files`: Ежечасно удаляет временные файлы, срок хранения которых истек.

- `process_translation_queue`: Каждые 5 секунд обрабатывает очередь заданий на перевод.
- `check_and_fix_stuck_jobs`: Каждые 5 минут разблокирует зависшие задания.
- `check_expired_subscriptions`: Ежедневно в 00:00 UTC проверяет истекшие подписки и переводит пользователей на бесплатный тариф.
- `send_upcoming_payment_notifications`: Ежедневно в 12:00 UTC отправляет уведомления о предстоящих платежах (за 3 дня).
- `process_auto_renewals`: Ежедневно в 12:00 UTC обрабатывает автоматические продления подписок.

2.2 КОНФИГУРАЦИЯ (`config.py`)

- **Управление настройками:** Используется `pydantic-settings` для управления конфигурацией приложения через переменные окружения и `.env` файл.
- **Структура `Settings`:** Класс `Settings` содержит все конфигурационные параметры, сгруппированные по назначению:
  - Общие настройки проекта (`PROJECT_NAME`, `SERVER_HOST`).
  - Настройки JWT (`SECRET_KEY`, `ALGORITHM`).
  - Настройки SMTP для отправки email.
  - Параметры для подключения к API различных LLM-провайдеров (DeepSeek, OpenAI, LMStudio).
  - Настройки безопасности (CORS, Allowed Hosts).
  - Настройки платежной системы ЮKassa.
  - Параметры очереди заданий и обработки переводов.
- **Валидация:** Для ключевых параметров реализованы кастомные валидаторы, обеспечивающие корректность данных при запуске приложения.

2.3 ПОДКЛЮЧЕНИЕ К БАЗЕ ДАННЫХ (`database.py`)

- **Движок SQLAlchemy:** Создается синхронный движок (`engine`) с использованием `QueuePool` для управления пулом соединений с PostgreSQL. Настроены параметры пула: `pool_size`, `max_overflow`, `pool_timeout`, `pool_recycle`.
- **Управление сессиями:**
  - `SessionLocal` является фабрикой для создания новых сессий.

- Функция-генератор `get_db` используется как зависимость (`Depends`) в FastAPI для предоставления сессии в эндпоинты. Она гарантирует, что сессия будет всегда закрыта (`db.close()`) после завершения запроса, предотвращая утечки соединений.
- **Базовая модель:** `declarative_base()` используется для создания базового класса `Base`, от которого наследуются все модели SQLAlchemy.
- **События SQLAlchemy:** Настроены обработчики событий для логирования моментов установки соединения, получения его из пула и возврата обратно.

2.4 Модуль перевода и валидации с помощью AI (`llm` и `translation`)

Для обеспечения высокого качества и надежности переводов в системе реализован самокорректирующийся механизм на базе `LangGraph`. Этот механизм не просто выполняет перевод, а создает конвейер проверки и исправления результата.

2.4.1 Цель механизма

Создать процесс перевода, который автоматически проверяет результат работы основной LLM и, в случае обнаружения недочетов (непереведенные фрагменты, пустой ответ), самостоятельно пытается исправить перевод.

2.4.2 Архитектура решения

Решение построено на базе графа состояний (`Stateful Graph`), реализованного в модуле `llm/graph_translator.py`.

1. Состояние графа (TranslationState): Это объект, который передается между узлами графа и содержит всю необходимую информацию: исходный текст, язык, домен, результат перевода, статус валидации и количество попыток.

2. Узлы графа:
   - **"Переводчик" (`translate_node`):** Первый узел, который выполняет первоначальный перевод текста с помощью выбранного LLM-провайдера.
   - **"Валидатор" (`validation_node`):** Ключевой узел, который проверяет качество полученного перевода. Он выполняет два типа проверок:
     - **Программная:** Быстрая проверка на очевидные ошибки (пустой результат, полное совпадение с оригиналом, слишком высокий процент общих слов).
     - **LLM-проверка:** Если программная проверка пройдена, для более сложных случаев привлекается LLM, которая оценивает полноту перевода.

– **"Корректор" (`correction_node`):** Этот узел активируется, если "Валидатор" обнаружил проблему. Он формирует новый, более строгий промпт для LLM, указывая на предыдущую ошибку, и запрашивает повторный, полный перевод.

3. Логика переходов:
   – После перевода управление всегда передается "Валидатору".
   – Если валидация успешна (`OK`), процесс завершается.
   – Если валидация провалена (`FAIL`) и количество попыток меньше 3, управление передается "Корректору".
   – После исправления управление снова возвращается к "Валидатору" для повторной проверки.
   – Если после 3 попыток исправления результат все еще неудовлетворительный, процесс завершается, и для сохранения целостности документа возвращается исходный текст.

2.4.3 Интеграция

– Механизм интегрирован в базовый класс `Translator` (`translation/translator.py`).
– Метод `translate_text` теперь не просто вызывает LLM, а запускает весь граф валидации.
– Система обеспечивает "изящную деградацию" (`graceful degradation`): в случае критического сбоя графа, она автоматически переключается на простой перевод без валидации, чтобы не прерывать процесс обработки файла.

2.4.4 Визуальная схема графа

На рисунке 1 представлена схема графа.

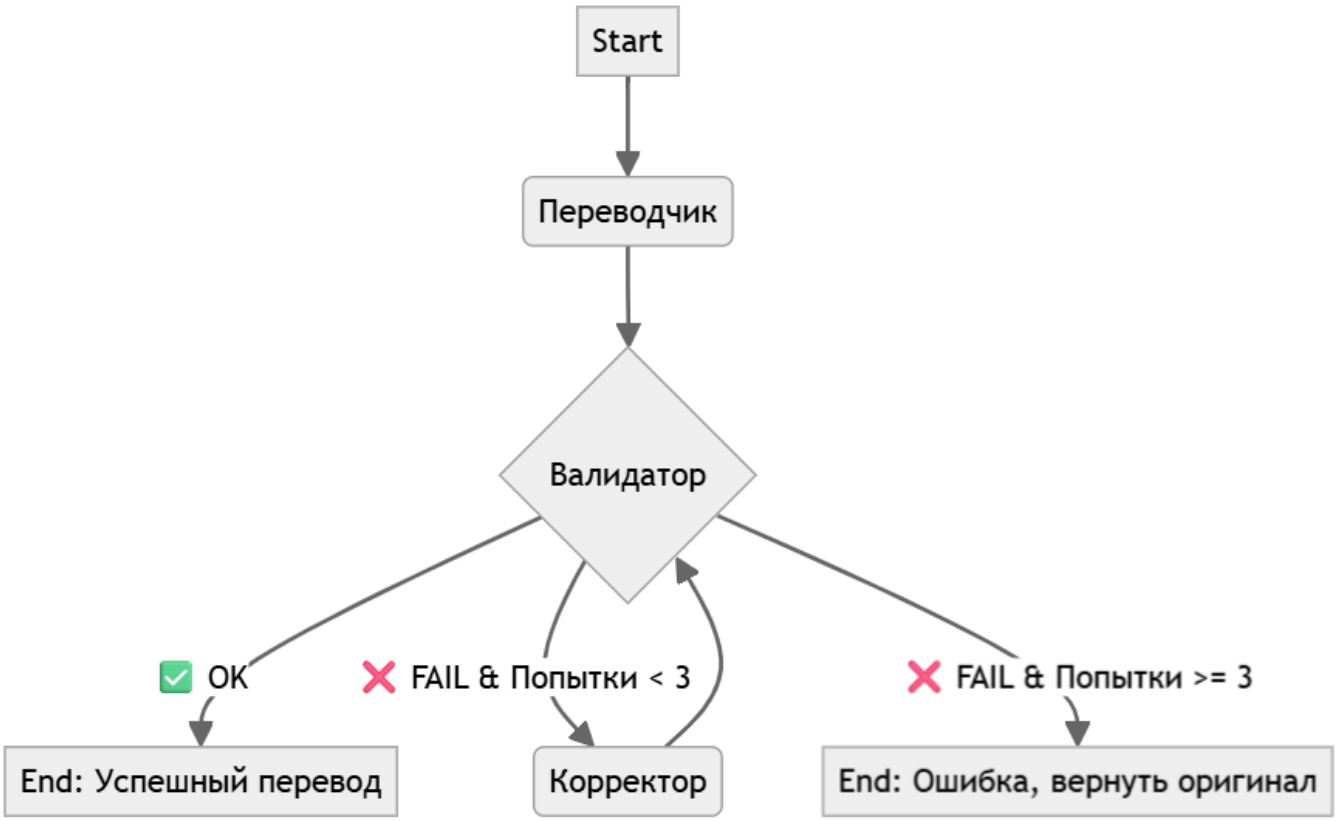

Рисунок 1 – Схема графа

2.4.5 Управление параллельной обработкой

Для оптимизации процесса массового перевода текстовых блоков в документах система использует параллельные запросы к LLM-провайдерам:

- **Универсальный параметр** `TRANSLATION_MAX_CONCURRENT`**:** Определяет максимальное количество одновременных запросов при переводе для любого провайдера. Это позволяет унифицировать управление нагрузкой независимо от выбранной LLM.
- **Реализация:** Во всех переводчиках по типам файлов (txt, docx, xlsx, pptx) текстовые блоки обрабатываются пакетами. Размер пакетов определяется именно настройкой `TRANSLATION_MAX_CONCURRENT`.
- **Конфигурация:** Параметр указывается в .env файле, по умолчанию имеет значение 20. Это значение выбрано как компромисс между скоростью перевода и нагрузкой на систему.

2.4.6 Диагностическое логирование

Система поддерживает два режима логирования, управляемых параметром `FULL_DIAGNOSTIC`:

– **DISABLED (по умолчанию):** Стандартное логирование только критических событий и ошибок.
– **ENABLED:** Полная диагностика с детальным логированием каждого переводимого блока.

При включенной диагностике (`FULL_DIAGNOSTIC=ENABLED`) логируется:

– Все блоки текста до и после перевода в формате "Было: ... / Стало: ...".
– Детальная информация о валидации и коррекции в GraphTranslator.
– Оригинальные тексты при возникновении ошибок FAIL_EMPTY и FAIL_UNCHANGED.
– Процесс принятия решений о повторных попытках перевода.

Это позволяет детально отслеживать качество переводов и диагностировать проблемы в продакшене.

Данный подход позволяет:

– Избежать перегрузки API LLM-провайдеров и получения ошибок по лимитам.
– Гибко масштабировать нагрузку в зависимости от мощности сервера и пропускной способности провайдеров.
– Единообразно управлять параллельностью для всех типов документов и провайдеров.

2.5 Система обеспечения надежности

Система использует отказоустойчивую архитектуру с сохранением всего состояния в базе данных для обеспечения надежности обработки файлов.

2.5.1 Архитектура очереди заданий

Система перешла от обработки файлов целиком в памяти к архитектуре с очередью заданий:

Поток обработки (Рисунок 2)**:**

```
Файл → Задание (Job) → Батчи (Batches) → Блоки (Blocks) → Очередь → Обработка → БД → Сборка → Готовый файл
```

Рисунок 2 – Поток обработки

Компоненты:

– **QueueManager** - управляет очередью заданий в БД с FIFO приоритетами.
– **JobProcessor** - обрабатывает задания, разбивает на блоки, управляет батчами.
– **BlockManager** - управляет отдельными блоками текста и их переводами.
– **Переводчики по типам файлов** - специализированные модули для каждого формата.

2.5.2 Механизмы надежности

1. **Транзакционность**: Все операции с БД выполняются в транзакциях.
2. **Блокировки**: Пессимистичные блокировки предотвращают конкурентную обработку.
3. **Контрольные точки**: Прогресс сохраняется после каждого батча.
4. **Повторные попытки**: Автоматические повторы при временных сбоях.
5. **Переключение провайдеров**: При ошибке LLM автоматически используется резервный.
6. **Мониторинг**: Автоматическое разблокирование зависших заданий.

2.5.3 Обработка сбоев

Система корректно обрабатывает различные типы сбоев:

- **Сбой LLM провайдера**: переключение на резервный.
- **Сбой сети**: повторная попытка с экспоненциальной задержкой.
- **Сбой процесса**: восстановление с последней контрольной точки.
- **Сбой БД**: ожидание восстановления соединения.
- **Сбой при сборке**: откат к последнему успешному состоянию.
- **Превышение лимитов**: корректная обработка и уведомление.

Подробное описание всех механизмов надежности доступно в документе: dbworking/dbworking_principles.md

2.6 ФАЙЛОВАЯ АРХИТЕКТУРА И УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ФАЙЛОВ

Система перешла на иерархическую архитектуру хранения файлов с автоматическим управлением их жизненным циклом для оптимизации использования дискового пространства и повышения безопасности.

2.6.1 Иерархическая структура хранения

Новая файловая структура представлена на рисунке 3:

```
files/
├── originals/
│   ├── 2025/01/06/                    # Год/Месяц/День создания
│   │   ├── 123.docx                   # ID файла = имя файла
│   │   └── translated_123.docx        # Переводы в той же директории
│   │   ├── 124.txt
│   │   └── translated_124.txt
│   └── 2025/01/07/
│       ├── 125.pptx
│       └── translated_125.pptx
└── archive/                           # Символическая ссылка на дешевое HDD хранилище
    └── 2025/01/06/
        └── translated_123.docx        # Архивированные переводы
```

Рисунок 3 – Иерархическая структура хранения

Ключевые принципы:

- **ID-based именование**: Файлы именуются по их ID в БД для безопасности.
- **Иерархическая структура**: Организация по дате создания `YYYY/MM/DD`.
- **Локальное размещение переводов**: Переводы сохраняются рядом с оригиналами.
- **Централизованное архивирование**: Автоматическое перемещение в дешевое хранилище.

2.6.2 Автоматическое управление жизненным циклом

Планировщик файлового архива:

- `cleanup_original_files()`: Ежедневно в 02:00 UTC:
  - Находит файлы со статусом `ready` старше 24 часов.
  - Удаляет оригинальные файлы с диска.
  - Обнуляет `file_path` в БД, сохраняя запись для истории.
  - Отправляет email администратору при ошибках.
- `archive_old_files()`: Ежедневно в 04:00 UTC:
  - Находит переводы старше 72 часов после `translated_at`.
  - Перемещает файлы из `files/originals/` в `files/archive/`.
  - Обновляет `translated_path` в БД на архивный путь.
  - Создает необходимые директории в архиве.
  - Логирует все операции и ошибки.

2.6.3 Утилитные функции (`files/utils.py`)

Функции генерации путей:

- `generate_file_path(file_id, extension, created_at)`: Создает иерархический путь для нового файла.
- `generate_translated_file_path_by_id(original_path, file_id)`: Создает путь для перевода рядом с оригиналом.
- `extract_file_id_from_path(file_path)`: Извлекает ID файла из пути.

Функции управления архивом:

- `move_to_archive(file_path)`: Перемещает файл в архивное хранилище.
- `is_file_archived(file_path)`: Проверяет, находится ли файл в архиве.
- `get_archive_path(original_path)`: Вычисляет архивный путь для файла.
- `cleanup_file_safely(file_path)`: Безопасное удаление с проверками.

2.6.4 Интеграция с переводчиками

Все переводчики (TXT, DOCX, PPTX, XLSX) обновлены для работы с новой архитектурой:

- Используют ID-based именование для переводов.
- Сохраняют переводы в той же директории что и оригиналы.
- Включают fallback механизм для обратной совместимости.
- Логируют пути файлов для диагностики.

2.6.5 Преимущества новой архитектуры

Безопасность:

- ID-based именование предотвращает утечку информации о содержимом файлов.
- Иерархическая структура усложняет несанкционированный доступ.

Производительность:

- Равномерное распределение файлов по директориям.
- Ускорение файловых операций за счет меньшего количества файлов в одной папке.
- Переводы рядом с оригиналами ускоряют доступ.

Управление ресурсами:

- Автоматическое освобождение места от оригинальных файлов.
- Перемещение старых переводов на дешевое хранилище.
- Сохранение истории файлов в БД без физических файлов.

Мониторинг:

– Email уведомления администратору о проблемах с файлами.
– Детальное логирование всех операций архивирования.
– Централизованное управление через планировщик задач.

2.7 Система автоматических платежей и мониторинга

Для предотвращения финансовых потерь от неконтролируемого использования LLM API реализована комплексная система мониторинга и автоматического управления подписками.

2.7.1 Автоматические продления подписок (`scheduler/tasks.py`)

Функция process_auto_renewals()**:**

– Ежедневно в 12:00 UTC находит подписки, истекающие завтра с включенным автопродлением.
– Создает рекуррентные платежи через ЮKassa API.
– При неудаче автоматически отключает автопродление и отправляет уведомление администратору.
– Ведет детальное логирование всех операций.

2.7.2 Мониторинг неудачных платежей (`payments/services.py`)

Обработка webhook от ЮKassa:

– Автоматическое обнаружение отклоненных рекуррентных платежей по статусу `canceled`.
– Определение рекуррентных платежей по метаданным `is_recurring: "true"`.
– Защитные меры при неудаче:
  – Отключение автопродления (`auto_renew = False`).
  – Удаление сохраненного способа оплаты.
  – Автоматический перевод на бесплатный тариф.
– Отправка уведомлений администратору и пользователю.

2.7.3 Email уведомления (`email/sync_sender.py`)

Типы уведомлений:

– `send_upcoming_payment_notification()`: За 3 дня до списания с деталями суммы и даты.
– `send_payment_success_notification()`: После успешной оплаты с информацией о подписке.

– `send_failed_payment_admin_notification()`: Детальное уведомление администратору о неудачном платеже.
– `send_payment_failed_user_notification()`: Инструкции пользователю по восстановлению подписки.

Дизайн уведомлений:

– Единая зеленая цветовая схема для всех типов уведомлений.
– Адаптивная верстка для корректного отображения на всех устройствах.
– Локализация на русский и английский языки.
– Ссылки на соответствующие разделы приложения.

2.7.4 Планировщик задач

Расписание выполнения:

– `process_auto_renewals`: 12:00 UTC ежедневно.
– `send_upcoming_payment_notifications`: 12:00 UTC ежедневно.
– `check_expired_subscriptions`: 00:00 UTC ежедневно.
– `process_recurring_subscriptions`: 03:00 UTC ежедневно.

Интеграция с APScheduler:

– Все задачи автоматически добавляются при старте приложения.
– Используется `AsyncIOScheduler` для неблокирующего выполнения.
– Автоматическое восстановление задач после перезапуска.

2.8 Система email уведомлений

2.8.1 Шаблоны уведомлений (`email/templates/`)

Шаблоны для платежей:

– `upcoming_payment_notification.html`: Уведомление о предстоящем платеже.
– `payment_success_notification.html`: Подтверждение успешной оплаты.
– `failed_payment_admin_notification.html`: Уведомление администратору о неудаче.
– `payment_failed_user_notification.html`: Инструкции пользователю.

Особенности шаблонов:

– Responsive дизайн для корректного отображения на мобильных устройствах.
– Единая цветовая схема с основным зеленым цветом (#52c41a).

– Поддержка переменных для персонализации.
– Ссылки на соответствующие разделы веб-приложения.

2.8.2 Отправка уведомлений

Синхронная отправка:

– Все email отправляются синхронно через SMTP.
– Автоматическая обработка ошибок подключения.
– Логирование всех попыток отправки.

Конфигурация SMTP:

– Поддержка различных SMTP провайдеров.
– Настройка через переменные окружения.
– Поддержка SSL/TLS шифрования.

3 МОДЕЛИ ДАННЫХ (СУЩНОСТИ)

Система использует SQLAlchemy для определения моделей данных, которые соответствуют таблицам в базе данных PostgreSQL. Pydantic используется для валидации данных, передаваемых через API.

3.1 АУТЕНТИФИКАЦИЯ И ПОЛЬЗОВАТЕЛИ (`auth`)

3.1.1 User (Таблица `users`)

Основная модель, представляющая пользователя системы.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `email`: Уникальный email пользователя (String, 255 символов).
  - `hashed_password`: Хеш пароля (String, 255 символов, nullable).
  - `is_verified`: Флаг, подтвержден ли email (Boolean, по умолчанию False).
  - `role`: Роль пользователя (`admin` или `user`, Enum, по умолчанию `user`).
  - `created_at`: Дата создания (DateTime с timezone).
  - `llm_provider`: Персональный LLM-провайдер для пользователя (String 50 символов, nullable).
  - `login_attempts`: Количество неудачных попыток входа (Integer, по умолчанию 0).
  - `is_blocked`: Флаг блокировки пользователя (Boolean, по умолчанию False).
  - `blocked_until`: Время до которого заблокирован (DateTime с timezone, nullable).
  - `last_failed_login`: Время последней неудачной попытки входа (DateTime с timezone, nullable).
  - `last_file_upload`: Время последней загрузки файла (DateTime с timezone, nullable).
  - `deletion_requested_at`: Время запроса удаления аккаунта (DateTime с timezone, nullable).
  - `email_hash_on_deletion`: Хеш email при удалении (String 255 символов, nullable).
  - `status_on_deletion`: Статус при удалении (String 100 символов, nullable).
  - `deleted_at`: Время удаления (DateTime с timezone, nullable).

- `temp_file_id`: Связь с временным файлом, загруженным до регистрации (Integer, ForeignKey).

- Связи:
  - `verification_tokens`: Один-ко-многим с токенами верификации.
  - `password_reset_tokens`: Один-ко-многим с токенами сброса пароля.
  - `account_deletion_tokens`: Один-ко-многим с токенами удаления аккаунта.
  - `subscriptions`: Один-ко-многим с подписками пользователя.
  - `usage_limit`: Один-к-одному с лимитами использования.
  - `promo_code_usages`: Один-ко-многим с использованиями промокодов.
  - `created_promo_codes`: Один-ко-многим с созданными промокодами (для админов).
  - `llm_usages`: Один-ко-многим с логами использования LLM.
  - `payments`: Один-ко-многим с платежами.
  - `temp_file`: Связь с временным файлом.

3.1.2 `VerificationToken`, `PasswordResetToken`, `AccountDeletionToken`

Модели для хранения временных токенов, используемых для подтверждения email, сброса пароля и удаления аккаунта. Содержат token, user_id и expires_at.

3.1.3 Схемы Pydantic (`auth.schemas`)

- `UserCreate`, `AdminCreate`: Для регистрации нового пользователя/администратора.
- `UserOut`: Для представления данных пользователя в API ответах (без пароля).
- `Token`: Для возврата JWT токена при логине.
- `UserLLMProviderSetting`: Для установки персонального LLM.

3.2 Файлы (`files`)

3.2.1 File (Таблица `files`)

Представляет файл, загруженный пользователем для перевода.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `original_name`: Исходное имя файла (String 255 символов).
  - `file_path`: Путь к оригинальному файлу на сервере (String 512 символов, уникальный).

  - `translated_path`: Путь к переведенному файлу (String 512 символов, nullable).
  - `language`: Целевой язык перевода (String 10 символов, динамическая валидация из БД).
  - `user_id`: Внешний ключ к users.id (Integer, nullable).
  - `status`: Текущий статус обработки (pending, processing, ready, error, Enum).
  - `created_at`: Время создания (DateTime с timezone).
  - `domain`: Предметная область (String 255 символов, nullable).
  - `translated_at`: Время завершения перевода (DateTime с timezone, nullable).
  - `error_message`: Сообщение об ошибке (String 512 символов, nullable).
  - `char_count_original`: Количество символов в исходном документе (Integer, nullable).
  - `is_temporary`: Флаг временного файла (Boolean, по умолчанию False).
  - `temp_id`: Уникальный ID для временных файлов (String 36 символов, nullable).
  - `llm_provider`: Провайдер LLM для перевода (String 50 символов, nullable).
  - `user_rating`: Оценка пользователя 0-3 (Integer, по умолчанию 0).
- Связи:
  - Принадлежит одному пользователю (`User`).
  - Один-ко-многим с заданиями на перевод (`TranslationJob`).

3.2.2 Схемы Pydantic (`files.schemas`)

- `FileOut`: Для представления данных о файле в API.
- `BulkUploadResponse`, `FileUploadError`: Для ответов при массовой загрузке.
- `DeleteFilesRequest`: Для запроса на удаление нескольких файлов.
- `TempUploadResponse`: Для ответа после загрузки временного файла.

3.3 Подписки и промокоды (`subscriptions`)

3.3.1 SubscriptionPlan (Таблица `subscription_plans`)

Описывает тарифный план.

- **Поля:** `name`, `description`, `price_monthly`, `price_yearly`, `currency_code`, `char_limit_monthly`, `is_active`, `is_public` и др.

3.3.2 UserSubscription (Таблица `user_subscriptions`)

Связывает пользователя с тарифным планом.

– **Поля:** `user_id`, `plan_id`, `start_date`, `end_date`, `status`, `auto_renew`, `yookassa_payment_method_id`.

3.3.3 UserUsageLimit (Таблица `user_usage_limits`)

Хранит информацию о текущем использовании лимитов символов пользователем.

– **Поля:** `user_id`, `free_tier_chars_used_today`, `paid_tier_chars_used_current_cycle`, `last_daily_reset_ts`.

3.3.4 `PromoCode` и `UserPromoCodeUsage`

Модели для создания и отслеживания использования промокодов.

3.3.5 Схемы Pydantic (`subscriptions.schemas`)

– Схемы `Create`, `Update`, `InDB` для каждой модели (`SubscriptionPlan`, `UserSubscription`, `PromoCode`).
– `UsageLimitsDetails`: Детализированная схема с информацией об остатках лимитов.

3.4 Платежи (`payments`)

3.4.1 Payment (Таблица `payments`)

Запись о финансовой операции.

– Поля:
  – `external_payment_id`: Уникальный ID платежа во внешней системе (ЮKassa).
  – `payment_system`: Платежная система (yookassa, Enum).
  – `user_id`, `user_subscription_id`: Связи с пользователем и его подпиской.
  – `amount`, `currency`, `status`: Сумма, валюта и статус платежа.
  – `confirmation_url`: URL для перехода на страницу оплаты.
  – `internal_metadata`, `external_metadata`: JSON-поля для хранения служебной информации.
– Связи:
  – Принадлежит одному пользователю (`User`) и одной подписке (`UserSubscription`).

3.4.2 Схемы Pydantic (`payments.schemas`)

– `PaymentInitiateRequest`, `PaymentInitiateResponse`: Для инициирования платежа с фронтенда.

– `YooKassaNotification`: Для десериализации вебхук-уведомлений от ЮKassa. Содержит вложенные схемы, полностью повторяющие структуру объектов от платежной системы (`YooKassaPaymentObject`, `YooKassaPaymentMethod` и т.д.).

3.5 Системные настройки и языки (`system`)

3.5.1 GlobalLLMProviderSetting (Таблица `global_llm_provider_settings`)

Глобальные настройки LLM-провайдера по умолчанию.

– Поля:
  – `id`: Первичный ключ (Integer).
  – `provider`: Название провайдера (`deepseek`, `openai`, `lmstudio`, `Enum`).
  – `created_at`, `updated_at`: Временные метки создания и обновления.

3.5.2 Language (Таблица `languages`)

Модель для управления поддерживаемыми языками системы.

– Поля:
  – `code`: Первичный ключ, код языка в формате ISO 639-1 (String, 2-5 символов).
  – `name_ru`: Название языка на русском языке (String).
  – `name_native`: Название языка на родном языке (String).
  – `direction`: Направление письма (`ltr` - слева направо, `rtl` - справа налево, Enum).
  – `enabled`: Флаг активности языка в системе (Boolean).
  – `created_at`, `updated_at`: Временные метки создания и обновления.
– Особенности:
  – Система поддерживает 18 языков: английский, русский, китайский, немецкий, французский, испанский, итальянский, португальский, японский, корейский, арабский, турецкий, хинди, нидерландский, польский, вьетнамский, тайский, индонезийский.
  – Языки с письмом справа налево (арабский) имеют `direction = 'rtl'`.
  – Все языки инициализируются как активные (`enabled = True`) при первом запуске системы.

3.5.3 Схемы Pydantic (`system.schemas`)

- `LLMProviderSettingIn`, `LLMProviderSettingOut`: Для управления глобальными настройками LLM.
- `LanguageCreate`, `LanguageUpdate`, `LanguageOut`: Для CRUD операций с языками.
- `LanguageToggleRequest`: Для переключения статуса активности языка.

3.6 Модели очереди переводов (`translation`)

3.6.1 TranslationJob (Таблица `translation_jobs`)

Представляет задание на перевод файла.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `file_id`: Внешний ключ к `files.id` (Integer, с индексом).
  - `status`: Статус задания (`pending`, `processing`, `completed`, `failed`, `cancelled`, Enum).
  - `source_language`: Исходный язык (String 10 символов).
  - `target_language`: Целевой язык (String 10 символов).
  - `domain`: Предметная область (String 255 символов, nullable).
  - `total_blocks`: Общее количество блоков (Integer, по умолчанию 0).
  - `processed_blocks`: Обработанные блоки (Integer, по умолчанию 0).
  - `failed_blocks`: Неудачные блоки (Integer, по умолчанию 0).
  - `created_at`: Время создания (DateTime с timezone).
  - `started_at`: Время начала обработки (DateTime с timezone, nullable).
  - `completed_at`: Время завершения (DateTime с timezone, nullable).
  - `error_message`: Сообщение об ошибке при сбое (Text, nullable).
  - `retry_count`: Количество повторных попыток (Integer, по умолчанию 0).
  - `job_metadata`: JSON метаданные для сборки файла (Text, nullable).
- Связи:
  - Принадлежит одному файлу (`File`).
  - Один-ко-многим с батчами (`TranslationBatch`).
  - Один-к-одному с записью в очереди (`TranslationQueue`).

3.6.2 TranslationBatch (Таблица `translation_batches`)

Представляет пакет блоков для параллельной обработки.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `job_id`: Внешний ключ к `translation_jobs.id` (Integer, с индексом).
  - `batch_number`: Порядковый номер батча (Integer).
  - `status`: Статус батча (`pending`, `processing`, `completed`, `failed`, Enum).
  - `created_at`: Время создания (DateTime с timezone).
  - `processed_at`: Время обработки (DateTime с timezone, nullable).
- Связи:
  - Принадлежит одному заданию (`TranslationJob`).
  - Один-ко-многим с блоками (`TranslationBlock`).

3.6.3 TranslationBlock (Таблица `translation_blocks`)

Представляет отдельный блок текста для перевода.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `batch_id`: Внешний ключ к `translation_batches.id` (Integer, с индексом).
  - `block_order`: Порядковый номер блока в батче (Integer).
  - `original_text`: Исходный текст блока (Text).
  - `translated_text`: Переведенный текст (Text, nullable).
  - `block_metadata`: JSON с метаданными (Text, nullable).
  - `status`: Статус блока (`pending`, `processing`, `completed`, `failed`, Enum).
  - `llm_provider`: Какой провайдер использовался (String 50 символов, nullable).
  - `retry_count`: Количество попыток (Integer, по умолчанию 0).
  - `created_at`: Время создания (DateTime с timezone).
  - `processed_at`: Время обработки (DateTime с timezone, nullable).
  - `error_message`: Сообщение об ошибке (Text, nullable).
- Связи:
  - Принадлежит одному батчу (`TranslationBatch`).

3.6.4 TranslationQueue (Таблица `translation_queue`)

Представляет очередь заданий для обработки.

- Поля:
  - `id`: Первичный ключ (Integer).
  - `job_id`: Внешний ключ к `translation_jobs.id` (Integer, уникальный).
  - `priority`: Приоритет (Integer, по умолчанию 100, с индексом).
  - `status`: Статус в очереди (`waiting`, `processing`, `completed`, `failed`, Enum).
  - `added_at`: Время добавления в очередь (DateTime с timezone, с индексом).
  - `started_at`: Время начала обработки (DateTime с timezone, nullable).
  - `completed_at`: Время завершения (DateTime с timezone, nullable).
- Связи:
  - Принадлежит одному заданию (`TranslationJob`).

4 API ENDPOINTS

API спроектировано по принципу REST и сгруппировано по функциональным модулям. Все эндпоинты, кроме системных (`/`, `/api/health`), доступны по префиксу `/api`.

4.1 Аутентификация и Управление Пользователями (`/api/auth`)

- `POST /register`: Регистрация нового пользователя. Проверяет, не был ли email недавно удален.
- `POST /login`: Аутентификация пользователя по email и паролю. Возвращает JWT токен.
- `GET /verify/email`: Верификация email по токену из письма. Запускает обработку временного файла, если он был загружен до регистрации.
- `GET /users/me`: Получение информации о текущем аутентифицированном пользователе.
- `POST /request-password-reset`: Запрос на сброс пароля. Отправляет письмо со ссылкой.
- `POST /reset-password`: Установка нового пароля по токену.
- `POST /users/me/delete-request`: Запрос на удаление своего аккаунта.
- `GET /confirm-delete/{token}`: Подтверждение удаления аккаунта по токену.
- `GET /login-attempts/{email}`: Получение количества неудачных попыток входа (публичный).
- **Административные эндпоинты:**
  - `GET /users`: Получение списка всех пользователей.
  - `DELETE /users/{user_id}`: Удаление пользователя.
  - `POST /admin/create`: Создание нового администратора.
  - `POST /admin/unblock/{user_id}`: Разблокировка пользователя.
  - `GET, PUT /admin/users/{user_id}/llm`: Управление персональным LLM-провайдером для пользователя.

4.2 Работа с Файлами (`/api/files`)

- `POST /upload`: Загрузка одного файла для перевода. Требует авторизации. Проверяет и списывает лимиты символов.
- `POST /bulk-upload`: Массовая загрузка до 10 файлов.

- `GET /`: Получение списка своих файлов (для пользователя) или файлов всех/конкретного пользователя (для администратора).
- `GET /{file_id}/download`: Скачивание переведенного файла.
- `POST /delete`: Массовое удаление файлов по их ID.
- `POST /upload-temp`: Загрузка временного файла для неавторизованного пользователя. Файл связывается с сессией и обрабатывается после регистрации.

4.3 Подписки и Промокоды (`/api/subscriptions`)

- Публичные эндпоинты:
  - `GET /plans`: Получение списка всех публичных тарифных планов.
  - `GET /plans/{plan_id}`: Получение детальной информации о конкретном плане.
  - `GET /promocodes/validate/{code_str}`: Проверка валидности промокода.
- Эндпоинты для аутентифицированных пользователей:
  - `GET /subscriptions/me/active`: Получение информации о своей текущей активной подписке и остатках лимитов.
  - `GET /subscriptions/me/history`: Получение истории всех своих подписок.
  - `POST /subscriptions/me/switch-to-free`: Переключение на бесплатный тариф.
- Административные эндпоинты:
  - `POST, GET /admin/plans`: Создание и получение списка всех тарифных планов.
  - `PUT, DELETE /admin/plans/{plan_id}`: Обновление и удаление тарифного плана.
  - `POST, GET /admin/promocodes`: Создание и получение списка всех промокодов.
  - `GET, PUT, DELETE /admin/promocodes/{promo_code_id}`: Управление конкретным промокодом.
  - `POST, GET /admin/users/{user_id}/subscriptions`: Назначение и просмотр подписок конкретного пользователя.
  - `PUT /admin/subscriptions/{subscription_id}`: Обновление конкретной подписки пользователя.

4.4 Платежи (`/api/payments`)

- `POST /initiate-payment`: Инициирование сессии оплаты через ЮKassa для выбранного тарифного плана. Возвращает confirmation_url для редиректа.
- `POST /yookassa/webhook`: Вебхук для приема уведомлений от ЮKassa о статусе платежей. Не требует авторизации и обрабатывает смену статуса подписки.

4.5 Системные (`/` и `/api/system`)

- `GET /`: Корневой эндпоинт для проверки доступности API.
- `GET /api/health`: Эндпоинт для проверки работоспособности бэкенда.
- `GET /api/languages`: Публичный эндпоинт для получения списка активных языков системы.
- **Административные эндпоинты:**
  - `GET, PUT /api/admin/settings/llm`: Просмотр и установка глобального LLM-провайдера по умолчанию.
  - `GET, POST /api/admin/settings/languages`: Получение списка всех языков и создание нового языка.
  - `PUT, DELETE /api/admin/settings/languages/{code}`: Обновление и удаление конкретного языка по коду.
  - `PATCH /api/admin/settings/languages/{code}/toggle`: Переключение статуса активности языка.

4.6 Управление очередью переводов (`/api/translation`)

- `GET /jobs`: Получение списка заданий на перевод (для администраторов).
- `GET /jobs/{job_id}`: Получение детальной информации о задании.
- `GET /jobs/{job_id}/progress`: Получение прогресса выполнения задания.
- `POST /jobs/{job_id}/cancel`: Отмена задания на перевод.
- `POST /jobs/{job_id}/retry`: Повторная попытка выполнения задания.
- `GET /batches/{batch_id}`: Получение информации о батче.
- `GET /blocks/{block_id}`: Получение информации о блоке.
- `GET /queue`: Просмотр текущей очереди заданий (для администраторов).
- `POST /queue/process`: Ручной запуск обработки очереди (для администраторов).

5 АРХИТЕКТУРА FRONTEND

Фронтенд-приложение является одностраничным (SPA), разработанным с использованием React и TypeScript.

5.1 Технологии и Библиотеки

- Сборщик: Vite.
- Язык: TypeScript.
- Фреймворк: React 19.
- UI-компоненты: Ant Design 5.x.
- Маршрутизация: React Router 7.x.
- HTTP-клиент: Axios.
- Интернационализация (i18n): `i18next` с плагинами для автоматического определения языка и загрузки переводов.
- Стилизация: CSS Modules и глобальные CSS-файлы.
- Линтинг: ESLint.

5.2 Структура Проекта (`src`)

- `/api`: Модули для взаимодействия с бэкенд API, сгруппированные по сущностям (`adminApi.ts`, `subscriptionApi.ts`). `axios.ts` содержит базовую конфигурацию Axios-инстанса и публичные API функции.
- `/components`: Переиспользуемые React-компоненты (`ProtectedRoute`, `LanguageSwitcher`, `ResponsiveTable`).
- `/context`: Контексты React для управления глобальным состоянием (аутентификация `AuthContext`, тема `ThemeContext`, уведомления `NotificationContext`).
- `/hooks`: Кастомные React-хуки для работы с API (`useLanguages.ts` для управления языками).
- `/layouts`: Компоненты-макеты страниц (например, `MainLayout`, который включает шапку, подвал и основное содержимое).
- `/locales`: JSON-файлы с переводами для разных языков (ru, en). Включают секцию `languageNames` с локализованными названиями всех поддерживаемых языков.
- `/pages`: Основные компоненты-страницы, соответствующие маршрутам в приложении (`TranslatorPage`, `LoginPage`, `AdminPage` и т.д.).

- `/services`: Логика для взаимодействия с API, отделенная от компонентов (например, `auth_service.ts`).
- `/theme`: Конфигурации тем (светлая `theme.ts` и темная `darkTheme.ts`) для Ant Design.
- `/types`: Файлы с определениями TypeScript-типов, используемых в приложении (`language.ts`, `file.ts`, `user.ts` и др.).
- `/utils`: Утилитарные функции (`languageUtils.ts` для форматирования названий языков с локализацией).

5.3 Управление состоянием

- **Локальное состояние:** Управляется с помощью хуков `useState` и `useReducer` внутри компонентов.
- **Глобальное состояние:** Реализовано через React Context API.
  - `AuthContext`: Хранит информацию о текущем пользователе, его роли и токене. Отвечает за логику входа, выхода и проверки статуса аутентификации. Предоставляет `isLoading` флаг, который используется для отображения прелоадера до окончания проверки аутентификации.
  - `ThemeContext`: Управляет переключением между светлой и темной темами.
  - `NotificationContext`: Предоставляет централизованный способ для отображения уведомлений (success, error, info) по всему приложению.

5.4 Маршрутизация (`App.tsx`)

- Используется `react-router-dom` для навигации.
- Все основные маршруты вложены в `MainLayout`, что обеспечивает единый вид (шапка/подвал) для большинства страниц.
- `ProtectedRoute`: Специальный компонент-обертка, который защищает маршруты. Он проверяет, аутентифицирован ли пользователь и имеет ли он необходимую роль (например, `admin`) для доступа к странице.

5.5 Интернационализация (`i18n.ts`)

- Поддерживаемые языки: Русский (`ru`), Английский (`en`).
- Язык по умолчанию: `ru`.

– `i18next-browser-languagedetector` используется для автоматического определения языка пользователя на основе URL, cookie или настроек браузера.
– Файлы с переводами загружаются асинхронно по мере необходимости.

6 КОМПОНЕНТЫ И СТРАНИЦЫ FRONTEND

В этом разделе описывается назначение каждой страницы (компонента из директории /pages).

6.1 Основные страницы

- `TranslatorPage.tsx`: Главная страница приложения. Позволяет пользователям загружать файлы для перевода.
  - **Функционал:** Выбор целевого языка и предметной области (домена). Загрузка файлов через Drag-and-Drop или диалоговое окно. Отображение списка загруженных файлов с возможностью удаления. Кнопка для запуска процесса перевода. Для авторизованных пользователей отображается таблица с историей последних 5 переводов, которая автоматически обновляется. Реализованы проверки на стороне клиента: ограничение на количество (10), размер одного файла (50 МБ) и суммарный размер (250 МБ), а также на тип файла. Имена файлов обрезаются до 50 символов.
- `TextTranslatePage.tsx`: Страница для перевода небольших фрагментов текста.
  - **Функционал:** Два текстовых поля для исходного и переведенного текста. Пользователь вводит текст, выбирает язык и нажимает "Перевести". Запрос отправляется на бэкенд как текстовый файл, и результат отображается во втором поле после завершения обработки.
- `TranslationHistoryPage.tsx`: Страница с полной историей переводов пользователя.
  - **Функционал:** Отображает таблицу со всеми файлами, которые когда-либо загружал пользователь. Позволяет скачивать готовые переводы и удалять файлы из истории. Статусы файлов обновляются автоматически.
- `SubscriptionPage.tsx`: Страница для просмотра и выбора тарифных планов.
  - **Функционал:** Отображает карточки доступных тарифных планов. Пользователь может выбрать период оплаты (месяц/год) и применить промокод. При нажатии на кнопку "Подписаться" инициируется процесс оплаты через ЮKassa.
- `UserProfilePage.tsx`: Страница профиля пользователя.
  - **Функционал:** Отображает информацию о пользователе (email, роль), детали текущей подписки и статистику использования лимитов символов (дневных и месячных). Предоставляет кнопки для запроса смены пароля и удаления аккаунта.

6.2 Аутентификация и доступ

- `LoginPage.tsx`: Страница входа в систему. Содержит форму с полями для email и пароля.
- `RegisterPage.tsx`: Страница регистрации. Содержит форму для ввода email, пароля и подтверждения пароля.
- `VerifyEmailPage.tsx`: Страница, на которую попадает пользователь после перехода по ссылке из письма для верификации. Она обрабатывает токен и сообщает о результате.
- `RequestPasswordResetPage.tsx`: Страница, где пользователь может ввести свой email, чтобы запросить ссылку для сброса пароля.
- `ResetPasswordPage.tsx`: Страница, на которую попадает пользователь по ссылке из письма для сброса пароля. Содержит форму для ввода нового пароля.

6.3 Администрирование

- `AdminPage.tsx`: Панель администратора, доступная только пользователям с ролью admin.
  - **Функционал:** Содержит вкладки для управления различными аспектами системы:
    - **Users Management:** Управление пользователями (просмотр, блокировка, смена роли).
    - **Files Management:** Управление файлами всех пользователей (просмотр, удаление).
    - **Plans Management:** Управление тарифными планами (создание, редактирование, удаление).
    - **Promo Codes Management:** Управление промокодами (создание, редактирование, деактивация).
    - **Language Management:** Управление поддерживаемыми языками системы (создание, редактирование, включение/выключение).
    - **Settings:** Управление глобальными и персональными настройками LLM-провайдеров.

6.4 Системные и служебные страницы

- `PaymentSuccessPage.tsx`: Страница, на которую пользователь перенаправляется после успешного завершения оплаты.

– `PaymentFailurePage.tsx`: Страница, на которую пользователь перенаправляется в случае ошибки или отмены платежа.
– `NotFoundPage.tsx`: Стандартная страница 404, отображается при переходе на несуществующий маршрут.

7 СИСТЕМА УПРАВЛЕНИЯ ЯЗЫКАМИ

Система управления языками представляет собой комплексное решение для динамического управления поддерживаемыми языками перевода. Она заменяет хардкодированные enum'ы на гибкую базу данных и предоставляет административный интерфейс для управления.

7.1 АРХИТЕКТУРА СИСТЕМЫ

7.1.1 Backend компоненты

- **Модель** `Language`: Хранит информацию о языках в базе данных PostgreSQL.
- **Сидер языков**: Автоматически инициализирует систему 18 языками при первом запуске.
- **API эндпоинты**: Публичные и административные эндпоинты для работы с языками.
- **CRUD операции**: Полный набор операций создания, чтения, обновления и удаления языков.

7.1.2 Frontend компоненты

- **Хук** `useLanguages`: Централизованное управление состоянием языков с кешированием.
- **Утилитарные функции**: Форматирование названий языков с локализацией.
- **Административный интерфейс**: Компонент `LanguagesManagement` для управления языками.
- **Интеграция**: Обновлены все компоненты для использования динамических языков.

7.2 ОСОБЕННОСТИ РЕАЛИЗАЦИИ

7.2.1 Локализация названий языков

Система поддерживает отображение названий языков в формате "Native Name (Localized Name)":

- **Русская локализация**: "English (Английский)", "中文 (Китайский)".
- **Английская локализация**: "Русский (Russian)", "中文 (Chinese)".
- **Автоматическое определение**: Если локализация недоступна, отображается только нативное название.

7.2.2 Управление активностью

– **Включение/выключение**: Администраторы могут деактивировать языки без их удаления.
– **Публичный API**: Возвращает только активные языки (`enabled = True`).
– **Административный API**: Предоставляет доступ ко всем языкам для управления.

7.2.3 Поддержка направления письма

– **LTR (Left-to-Right)**: Большинство языков (английский, русский, китайский и др.).
– **RTL (Right-to-Left)**: Арабский.
– **Будущее развитие**: Возможность использования для корректного отображения UI.

7.3 Поддерживаемые языки

Система изначально поддерживает 18 языков (Таблица 1):

Таблица 1 – Список поддерживаемых языков (с локализацией)

Код	Название (нативное)	Направление	Локализация (RU)	Локализация (EN)
en	English	LTR	Английский	English
ru	Русский	LTR	Русский	Russian
zh	中文	LTR	Китайский	Chinese
de	Deutsch	LTR	Немецкий	German
fr	Français	LTR	Французский	French
es	Español	LTR	Испанский	Spanish
it	Italiano	LTR	Итальянский	Italian
pt	Português	LTR	Португальский	Portuguese
ja	日本語	LTR	Японский	Japanese
ko	한국어	LTR	Корейский	Korean
ar	العربية	RTL	Арабский	Arabic
tr	Türkçe	LTR	Турецкий	Turkish
hi	हिन्दी	LTR	Хинди	Hindi
nl	Nederlands	LTR	Нидерландский	Dutch
pl	Polski	LTR	Польский	Polish
vi	Tiếng Việt	LTR	Вьетнамский	Vietnamese
th	ภาษาไทย	LTR	Тайский	Thai
id	Bahasa Indonesia	LTR	Индонезийский	Indonesian

7.4 Преимущества новой системы

7.4.1 Гибкость

- **Динамическое управление**: Добавление новых языков без изменения кода.
- **Быстрое включение/выключение**: Моментальная активация/деактивация языков.
- **Централизованное управление**: Единая точка управления всеми языками.

7.4.2 Масштабируемость

- **Легкое расширение**: Простое добавление новых языков через админ-панель.
- **Поддержка метаданных**: Возможность добавления дополнительных полей в будущем.
- **API-first подход**: Готовность к интеграции с внешними системами.

7.4.3 Пользовательский опыт

- **Локализованные названия**: Понятные названия языков для пользователей.
- **Консистентность**: Единообразное отображение во всех компонентах.
- **Производительность**: Кеширование и оптимизированные запросы.

8 ИНФРАСТРУКТУРА И РАЗВЕРТЫВАНИЕ

Проект полностью контейнеризирован с использованием Docker и Docker Compose, что обеспечивает легкость развертывания и масштабирования.

8.1 Сервисы (`docker-compose.yml`)

Оркестрация проекта осуществляется через `docker-compose.yml`, который определяет четыре основных сервиса:

- `backend`:
  - **Назначение:** Основное FastAPI приложение.
  - **Сборка:** Собирается из `backend/Dockerfile`.
  - **Контейнер:** `backend`.
  - **Порты:** Expose 8000 (доступен внутри Docker-сети).
  - **Зависимости:** Запускается после старта сервиса `db`.
  - **Переменные окружения:** Загружаются из `.env` файла. Дополнительно установлены `TZ` и `SOCKS_PROXY_URL`.
  - **Volumes:**
    - `./backend:/app` - исходный код для разработки.
    - `static_volume:/app/static` - статические файлы.
    - `./backend/files:${FILES_DIR}` - загруженные файлы.
  - **Restart:** `unless-stopped`.
- `frontend`:
  - **Назначение:** Сборщик статических файлов фронтенда.
  - **Сборка:** Собирается из `frontend/Dockerfile`.
  - **Контейнер:** `frontend_builder`.
  - **Restart:** `no` (запускается один раз для сборки).
  - **Volumes:** `frontend_dist:/app/dist` - собранные файлы.
- `db`:
  - **Назначение:** База данных PostgreSQL.
  - **Образ:** `postgres:16`.
  - **Контейнер:** `postgres_db`.
  - **Порты:** `5432:5432` (доступен на хосте).
  - **Переменные окружения:** `POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_DB`, `TZ`, `PGTZ`.

- **Volumes:** `postgres_data:/var/lib/postgresql/data`.
- **Restart:** `unless-stopped`.

- `nginx`:
  - **Назначение:** Веб-сервер и обратный прокси.
  - **Сборка:** Собирается из `nginx/Dockerfile`.
  - **Контейнер:** `nginx`.
  - **Порты:** `80:80` (основной порт доступа).
  - **Зависимости:**
    - `frontend`: `service_completed_successfully`.
    - `backend`: `service_started`.
  - Volumes:
    - `frontend_dist:/var/www/frontend` - статические файлы React.
    - `./marketing_pages:/var/www/marketing` - маркетинговые страницы.
    - `./nginx/default.conf.template:/etc/nginx/templates/default.conf.template` - шаблон конфигурации.
  - **Переменные окружения:** `SERVER_DOMAIN`, `BACKEND_HOST`, `BACKEND_PORT`, `NGINX_MAX_TOTAL_SIZE_MB`, `TZ`.
  - **Restart:** `unless-stopped`.

8.2 Конфигурация Nginx (`nginx/default.conf.template`)

Nginx выступает в роли единой точки входа в приложение и использует шаблонизацию для подстановки переменных окружения.

- **Обратный прокси:** Все запросы, начинающиеся с `/api/`, перенаправляются на сервис `backend` (на `http://${BACKEND_HOST}:${BACKEND_PORT}`). Устанавливаются необходимые заголовки для корректной работы бэкенда.
- **Обслуживание статики:**
  - Запросы к корневому пути (`/`) обслуживаются из директории `/var/www/marketing`, отдавая `index.html` маркетинговой страницы.
  - Все остальные запросы обслуживаются из директории `/var/www/frontend`, где лежат собранные файлы React-приложения. `try_files` настроен для корректной работы SPA роутинга.
- **Лимиты:** `client_max_body_size` настраивается через переменную `${NGINX_MAX_TOTAL_SIZE_MB}m` для поддержки массовой загрузки файлов.

– **Переменные окружения:** Конфигурация использует `envsubst` для подстановки переменных при старте контейнера.

9 СИСТЕМА АВТОМАТИЧЕСКИХ ПЛАТЕЖЕЙ И МОНИТОРИНГА

Для предотвращения финансовых потерь от неконтролируемого использования LLM API реализована комплексная система мониторинга и автоматического управления подписками.

9.1 Автоматические продления подписок

9.1.1 Архитектура автопродлений (`scheduler/tasks.py`)

Функция `process_auto_renewals()`:

- Выполняется ежедневно в 12:00 UTC через APScheduler.
- Находит подписки, истекающие завтра с включенным автопродлением (`auto_renew = True`).
- Создает рекуррентные платежи через ЮKassa API с использованием сохраненного способа оплаты.
- При неудаче автоматически отключает автопродление и отправляет уведомление администратору.
- Ведет детальное логирование всех операций для мониторинга.

Создание рекуррентных платежей:

- Использует сохраненный `yookassa_payment_method_id` из подписки.
- Устанавливает метаданные `is_recurring: "true"` для идентификации типа платеж.
- Автоматически рассчитывает сумму на основе тарифного плана.
- Создает запись в таблице `payments` для отслеживания.

9.1.2 Интеграция с ЮKassa

API взаимодействие:

- Использует `create_recurring_yookassa_payment()` для создания рекуррентных платежей.
- Передает сохраненный ID способа оплаты (`payment_method_id`).
- Устанавливает описание платежа с указанием периода подписки.
- Обрабатывает ошибки API и уведомляет администратора.

9.2 Мониторинг неудачных платежей

9.2.1 Webhook обработка (`payments/services.py`)

Функция `process_yookassa_notification()`:

- Обрабатывает все уведомления от ЮKassa о смене статуса платежей.
- Автоматически обнаруживает отклоненные рекуррентные платежи по статусу `canceled`.
- Определяет рекуррентные платежи по метаданным `is_recurring: "true"`.
- Применяет защитные меры при обнаружении неудачного автопродления.

Защитные механизмы:

- Отключение автопродления (`auto_renew = False`).
- Удаление сохраненного способа оплаты (`yookassa_payment_method_id = None`).
- Автоматический перевод пользователя на бесплатный тариф.
- Отправка детальных уведомлений администратору и пользователю.

9.2.2 Обработка различных сценариев

Типы обрабатываемых ошибок:

- Недостаток средств на карте.
- Истечение срока действия карты.
- Блокировка карты банком.
- Технические ошибки платежной системы.
- Отклонение банком-эмитентом.

9.3 Email уведомления

9.3.1 Типы уведомлений (`email/sync_sender.py`)

Уведомления о платежах:

- `send_upcoming_payment_notification()`: За 3 дня до автоматического списания.
- `send_payment_success_notification()`: Подтверждение успешной оплаты.
- `send_failed_payment_admin_notification()`: Детальное уведомление администратору.
- `send_payment_failed_user_notification()`: Инструкции пользователю по восстановлению.

Содержание уведомлений:

- Детали платежа (сумма, дата, тарифный план).
- Информация о пользователе и подписке.
- Причина неудачи (для администратора).
- Инструкции по восстановлению (для пользователя).
- Ссылки на соответствующие разделы приложения.

9.3.2 Шаблоны уведомлений (`email/templates/`)

Дизайн и верстка:

- Единая зеленая цветовая схема (#52c41a) для всех уведомлений.
- Responsive дизайн для корректного отображения на мобильных устройствах.
- Поддержка переменных для персонализации контента.
- Локализация на русский и английский языки.

Шаблоны для различных сценариев:

- `upcoming_payment_notification.html`: Предупреждение о предстоящем списании.
- `payment_success_notification.html`: Подтверждение успешной оплаты.
- `failed_payment_admin_notification.html`: Уведомление администратору (красная тема).
- `payment_failed_user_notification.html`: Инструкции пользователю (желтая тема).

9.4 Планировщик задач (APScheduler)

9.4.1 Конфигурация планировщика

Расписание выполнения:

- `process_auto_renewals`: 12:00 UTC ежедневно - обработка автопродлений.
- `send_upcoming_payment_notifications`: 12:00 UTC ежедневно - уведомления о предстоящих платежах.
- `check_expired_subscriptions`: 00:00 UTC ежедневно - проверка истекших подписок.
- `process_recurring_subscriptions`: 03:00 UTC ежедневно - обработка рекуррентных подписок.

Интеграция с приложением:

– Все задачи автоматически добавляются при старте приложения в `main.py`.
– Используется `AsyncIOScheduler` для неблокирующего выполнения.
– Автоматическое восстановление задач после перезапуска контейнера.
– Логирование запуска и завершения каждой задачи.

9.4.2 Мониторинг выполнения

Логирование:

– Детальные логи каждой задачи с временными метками.
– Информация о количестве обработанных записей.
– Ошибки и исключения с полным стеком вызовов.
– Статистика выполнения для анализа производительности.

9.5 Интеграция с фронтендом

9.5.1 Управление автопродлением

Компонент `AutoRenewalSettings`:

– Переключатель для включения/выключения автопродления.
– Отображение информации о сохраненном способе оплаты.
– Предупреждения о предстоящих списаниях.
– Возможность удаления сохраненного способа оплаты.

Интеграция в профиль пользователя:

– Отображение статуса автопродления в `UserProfilePage`.
– Уведомления о проблемах с платежами.
– Ссылки на страницу подписки для восстановления.

9.5.2 Отображение статуса платежей

Индикаторы состояния:

– Успешные платежи с зеленой индикацией.
– Проблемы с платежами с красной индикацией.
– Предстоящие платежи с желтой индикацией.
– Отключенное автопродление с серой индикацией.

10 БЕЗОПАСНОСТЬ И ЗАЩИТА ОТ ФИНАНСОВЫХ ПОТЕРЬ

10.1 Многоуровневая защита

10.1.1 Автоматические защитные механизмы

Немедленные действия при неудачных платежах:

– Автоматическое отключение автопродления для предотвращения повторных попыток.
– Удаление сохраненного способа оплаты для исключения неконтролируемых списаний.
– Перевод пользователя на бесплатный тариф для ограничения расходов на LLM.
– Блокировка возможности превышения лимитов бесплатного тарифа.

10.1.2 Уведомления и мониторинг

Система раннего предупреждения:

– Уведомления о предстоящих платежах за 3 дня до списания.
– Мгновенные уведомления администратору о любых проблемах с платежами.
– Детальная информация о причинах неудач для быстрого реагирования.
– Автоматические отчеты о финансовом состоянии системы.

10.2 Контроль расходов на LLM

10.2.1 Лимитирование использования

Жесткие ограничения:

– Проверка лимитов перед каждым запросом к LLM.
– Невозможность превышения установленных лимитов.
– Автоматическая остановка обработки при достижении лимитов.
– Уведомления пользователей о приближении к лимитам.

10.2.2 Мониторинг потребления

Отслеживание использования:

– Реальное время отслеживания расхода символов.
– Ежедневные и месячные отчеты по использованию.
– Анализ трендов потребления для прогнозирования.
– Алерты при аномальном росте потребления.

10.3 Административный контроль

10.3.1 Панель мониторинга

Инструменты администратора:

– Просмотр всех активных подписок и их статусов.
– Мониторинг неудачных платежей в реальном времени.
– Возможность ручного отключения автопродления для проблемных аккаунтов.
– Статистика по финансовым потерям и их предотвращению.

10.3.2 Экстренные меры

Возможности быстрого реагирования:

– Массовое отключение автопродления при критических проблемах.
– Временная блокировка новых подписок при технических сбоях.
– Ручной перевод пользователей на бесплатные тарифы.
– Экстренное отключение LLM провайдеров при превышении бюджетов.