Техническая документация проекта

1. Обзор проекта

Данный проект представляет собой интегрированную систему, состоящую из двух основных компонентов:

- **llm_module** — модуль работы с языковыми моделями (LLM) на ветке development
- **frontend** — веб-интерфейс приложения

Система предназначена для взаимодействия с языковыми моделями через унифицированный API.

2. Архитектура системы

2.1 Общая структура

```
Project
├── llm_module/ # Backend модуль для работы с LLM
│   └── development/ # Ветка разработки
└── frontend/ # Frontend приложение
```

2.2 Компоненты системы

LLM Module (backend)

- Основной модуль для интеграции и управления языковыми моделями
- Расположен на ветке `development`
- Предоставляет API для frontend приложения

Frontend

- Веб-интерфейс приложения
- Взаимодействует с LLM Module через API
- Разработана с использованием современных веб-технологий

3. Разработчики

Проект разрабатывается двумя контрибьюторами:

- **lolikatena** — GitHub профиль
- **Pe4en1e** — GitHub профиль

4. Описание компонентов

4.1 LLM Module (development branch)

Назначение: Основной модуль для работы с языковыми моделями

Основные функции:

- Интеграция с различными LLM провайдерами
- Управление запросами к моделям
- Обработка и трансформация данных
- Кеширование результатов (при необходимости)
- Обработка ошибок и восстановление после сбоев
- Валидация входных и выходных данных

Технологический стек:

- Python 3.8+ (основной язык)
- FastAPI или Flask (веб-фреймворк)
- Библиотеки для работы с LLM (openai, langchain, huggingface и т.д.)
- PostgreSQL или MongoDB (база данных)
- Redis (для кеширования)
- Docker (контейнеризация)

Архитектурные слои:

- API Layer — HTTP endpoints для взаимодействия с frontend
- Service Layer — основная бизнес-логика работы с моделями
- Data Layer — работа с базой данных и кешем
- Integration Layer — интеграция с внешними LLM провайдерами

Основные модули:

- `main.py` — точка входа приложения
- `api/routes.py` — определение API endpoints
- `services/` — бизнес-логика
- `models/` — структуры данных и модели БД
- `utils/` — вспомогательные функции
- `config.py` — конфигурация приложения

Безопасность:

- Валидация всех входящих данных
- Обработка исключений с информативными сообщениями об ошибках
- Rate limiting для предотвращения злоупотреблений
- Шифрование чувствительных данных
- JWT или OAuth2 для аутентификации

4.2 Frontend

Назначение: Веб-интерфейс для взаимодействия с системой

Основные функции:

- Визуализация интерфейса пользователя
- Отправка запросов к LLM Module
- Отображение результатов обработки
- Управление сеансами пользователя
- Локальное хранилище истории запросов
- Форматирование и подсветка синтаксиса результатов
- Управление профилем пользователя

Технологический стек:

- React.js или Vue.js (фреймворк)
- JavaScript/TypeScript
- Redux или Vuex (управление состоянием)
- Axios (HTTP клиент)
- CSS-in-JS (Styled Components, Tailwind CSS)
- Webpack или Vite (сборщик)
- Jest или Vitest (тестирование)

Структура проекта:

- `src/components/` — переиспользуемые компоненты UI
- `src/pages/` — страницы приложения
- `src/services/` — API сервисы
- `src/store/` — управление состоянием
- `src/utils/` — вспомогательные функции
- `src/styles/` — глобальные стили
- `public/` — статические ресурсы

Компоненты пользовательского интерфейса:

- Chat Interface — основное окно для взаимодействия с моделью
- Input Panel — поле ввода запроса с кнопками управления
- Message Display — отображение истории сообщений
- Settings Panel — настройки параметров модели
- User Profile — профиль и настройки пользователя
- Navigation Menu — меню навигации приложения
- Error Handler — отображение и обработка ошибок

Особенности:

- Адаптивный дизайн для мобильных и десктопных устройств
- Темная и светлая тема оформления
- Real-time обновления при получении ответа от сервера
- Оффлайн поддержка базового функционала
- Интерактивные компоненты с плавными анимациями

5. API интеграция

5.1 Взаимодействие компонентов

Frontend отправляет запросы к LLM Module через REST API или WebSocket соединение. Основной поток данных:

1. Пользователь вводит запрос в UI
2. Frontend отправляет запрос к LLM Module
3. LLM Module обрабатывает запрос с помощью языковой модели
4. Результат возвращается в Frontend
5. Frontend отображает результат пользователю

5.2 REST API Endpoints

Основные endpoints LLM Module:

Метод	Endpoint	Назначение	Описание
POST	/api/v1/chat	Отправить запрос	Отправляет запрос к LLM модели
GET	/api/v1/chat/{id}	Получить результат	Получает результат по ID запроса
GET	/api/v1/history	История чата	Получает историю запросов пользователя
DELETE	/api/v1/chat/{id}	Удалить запрос	Удаляет запрос из истории
POST	/api/v1/auth/login	Авторизация	Вход в систему
POST	/api/v1/auth/logout	Выход	Выход из системы
GET	/api/v1/models	Список моделей	Получает доступные модели

Table 1: REST API Endpoints LLM Module

5.3 Формат данных

Пример запроса:

```
{
"message": "Объясни принципы машинного обучения",
"model": "gpt-4",
"parameters": {
"temperature": 0.7,
"max_tokens": 2000
},
"session_id": "user_session_123"
}
```

Пример ответа:

```
{
"id": "request_456",
"status": "success",
"message": "Машинное обучение...",
"model": "gpt-4",
"processing_time": 2.34,
"tokens_used": 150,
"timestamp": "2025-12-11T20:49:00Z"
}
```

5.4 WebSocket соединение

Для потоковой передачи данных в реальном времени используется WebSocket:

- **URL:** `wss://api.example.com/ws/chat`
- **Сообщения:** JSON формат
- **Аутентификация:** Bearer token в headers
- **Переподключение:** Автоматическое с экспоненциальной задержкой

5.5 Обработка ошибок

Код	Ошибка	Описание
400	Bad Request	Некорректный формат запроса
401	Unauthorized	Требуется аутентификация
403	Forbidden	Доступ запрещён
404	Not Found	Ресурс не найден
429	Too Many Requests	Превышен лимит запросов
500	Server Error	Внутренняя ошибка сервера
503	Service Unavailable	Сервис недоступен

Table 2: HTTP коды ошибок

6. Развёртывание

6.1 Требования к окружению

Для LLM Module:

- Python 3.8 или выше
- pip (менеджер пакетов Python)
- Virtual environment (рекомендуется)
- Git для управления версиями
- Docker и Docker Compose (для контейнеризации)
- Доступ к API ключам провайдеров LLM
- Минимум 4GB RAM и 2 CPU ядра
- PostgreSQL 12+ (для хранения данных)
- Redis 6+ (для кеширования)

Для Frontend:

- Node.js 16.0+ и npm/yarn
- Git для управления версиями
- Современный браузер (Chrome, Firefox, Safari, Edge)
- Доступ к Backend API
- Webpack или Vite (идёт в комплекте)

Для обеих частей:

- Доступ в интернет
- HTTPS поддержка для production среды

6.2 Инструкции по установке (локальная разработка)

LLM Module:

1. Клонирование репозитория:
2. Создание виртуального окружения:
3. Установка зависимостей:
4. Конфигурация переменных окружения:
5. Инициализация базы данных:
6. Запуск сервера:

Frontend:

1. Клонирование репозитория:
2. Установка зависимостей:
3. Конфигурация окружения:
4. Запуск development сервера:

6.3 Docker развёртывание

Docker Compose для полного стека:

\begin{verbatim}
version: '3.8'
services:
backend:
build: ./llm_module
ports:
- "8000:8000"
environment:
- DATABASE_URL=postgresql://user:pass@db:5432/llm_db
- REDIS_URL=redis://redis:6379
depends_on:
- db
- redis
frontend:
build: ./frontend
ports:
- "80:3000"
environment:
- REACT_APP_API_URL=http://backend:8000
db:
image: postgres:15
environment:
- POSTGRES_DB=llm_db
- POSTGRES_USER=user
- POSTGRES_PASSWORD=pass
volumes:
- db_data:/var/lib/postgresql/data
redis:
image: redis:7-alpine
volumes:
db_data:
\end{verbatim}

Запуск с Docker Compose:

6.4 Production развёртывание

Рекомендуемая архитектура:

- Nginx — обратный прокси и балансировщик нагрузки
- Gunicorn — WSGI сервер для Python
- PM2 или systemd — управление процессами
- SSL/TLS сертификаты от Let's Encrypt
- CDN для статических файлов
- Мониторинг и логирование (ELK Stack, Prometheus)

Чек-лист перед production:

1. Установить все зависимости
2. Запустить тесты: `npm test` и `pytest`
3. Собрать production сборку: `npm run build`

4. Настроить переменные окружения
5. Активировать HTTPS и CORS политики
6. Настроить резервное копирование БД
7. Установить мониторинг и алерты
8. Создать документацию по процедурам восстановления

7. Статус проекта

- **Статус разработки:** В активной разработке
- **Ветка разработки:** development
- **Количество контрибьюторов:** 2
- **Релизы:** Не опубликованы
- **Пакеты:** Не опубликованы
- **Форки:** 1

7. Тестирование

7.1 Unit тесты (Backend)

Framework: pytest
\begin{verbatim}
Запуск тестов

pytest tests/
С покрытием кода

pytest --cov=app tests/
Конкретный тест

pytest tests/test_api.py::test_chat_endpoint
\end{verbatim}
Примеры тестов:
- Тесты API endpoints
- Тесты сервисов обработки LLM
- Тесты валидации входных данных
- Тесты базы данных

7.2 Unit тесты (Frontend)

Framework: Jest или Vitest
\begin{verbatim}

Запуск тестов

npm test

Watch режим

npm test -- --watch

С покрытием

npm test -- --coverage
\end{verbatim}
Примеры тестов:

- Тесты компонентов React
- Тесты Redux actions и reducers
- Тесты API сервисов
- Snapshot тесты

7.3 Integration тесты

- End-to-end тесты с Cypress или Playwright
- Тесты взаимодействия Frontend и Backend
- Тесты в Docker окружении

7.4 Performance тесты

- Load testing с помощью Apache JMeter или Locust
- Анализ времени отклика API
- Мониторинг использования памяти и CPU

8. Безопасность и лучшие практики

8.1 Security рекомендации

1. Использовать HTTPS для всех соединений
2. Валидировать и санитизировать все входные данные
3. Использовать параметризованные SQL запросы для предотвращения SQL injection
4. Реализовать rate limiting и защиту от DDoS
5. Хранить пароли с использованием bcrypt или аналога
6. Регулярно обновлять зависимости
7. Использовать CORS политики правильно
8. Логировать все действия безопасности
9. Проводить security аудиты
10. Получить сертификат безопасности (SSL/TLS)

8.2 Best practices разработки

- Code review перед merge в production
- Использовать version control и semantic versioning
- Поддерживать код в чистом виде (linting, formatting)
- Документировать код и функции
- Следовать PEP 8 (Python) и ESLint (JavaScript) стилям
- Использовать type hints (Python) и TypeScript
- Разделять логику на слои (API, Service, Data)
- Избегать hardcoded значений — использовать конфиг файлы
- Логировать важные события и ошибки
- Регулярно рефакторить код

9. Дополнительная информация

9.1 Репозитории

- LLM Module: https://github.com/pp2404-1/llm_module
- Frontend: https://github.com/pp2404-1/frontend

9.2 Рекомендации для дальнейшей разработки

1. Добавить подробные README файлы в оба репозитория с инструкциями по установке
2. Настроить CI/CD pipeline для автоматизации тестирования и развёртывания
3. Документировать API endpoints с помощью Swagger/OpenAPI (автогенерация из кода)
4. Добавить comprehensive unit и integration тесты для обоих компонентов
5. Настроить логирование и мониторинг в production (ELK, Prometheus, Grafana)
6. Реализовать graceful shutdown и обработку сигналов
7. Добавить OAuth2/JWT аутентификацию и авторизацию
8. Внедрить кеширование на уровне API и БД
9. Оптимизировать производительность (database indexing, query optimization)
10. Добавить analytics и user tracking (не нарушая privacy)
11. Создать документ по архитектурным решениям (ADR)
12. Настроить автоматическое резервное копирование БД
13. Реализовать версионирование API для обратной совместимости
14. Добавить Rate Limiting и Quotas для API
15. Документировать процедуры развёртывания и откатов

9.3 CI/CD Pipeline пример

GitHub Actions workflow:
\begin{verbatim}
name: CI/CD Pipeline
on: [push, pull_request]
jobs:
test:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2

- uses: actions/setup-python@v2
- run: pip install -r requirements.txt
- run: pytest tests/
build:
runs-on: ubuntu-latest
needs: test
steps:
- uses: actions/checkout@v2
- run: docker build -t llm-module .
- run: docker push registry.example.com/llm-module
deploy:
runs-on: ubuntu-latest
needs: build
if: github.ref == 'refs/heads/main'
steps:
- run: |
ssh deploy@server.com
cd /app && docker pull latest
docker-compose up -d
\end{verbatim}

9.4 Мониторинг и логирование

Рекомендуемые инструменты:

Задача	Инструмент	Описание
Логирование	ELK Stack (Elasticsearch)	Централизованное хранилище логов
Мониторинг метрик	Prometheus + Grafana	Сбор и визуализация метрик
APM	New Relic или Datadog	Application Performance Monitoring
Error tracking	Sentry	Отслеживание ошибок в production
Алерты	PagerDuty или Opsgenie	Оповещения команды о проблемах

Table 3: Инструменты мониторинга

10. Контакты и поддержка

Для вопросов и предложений по проекту рекомендуется:

- Обратиться к разработчикам через GitHub Issues в соответствующих репозиториях
- Создать Pull Request с исправлениями или улучшениями
- Связаться с разработчиками напрямую через их GitHub профили
- Оставить feedback в разделе Discussions репозитория

Разработчики проекта:

- lolikatena: https://github.com/lolikatena
- Pe4en1e: https://github.com/Pe4en1e

11. Глоссарий

- **LLM** (Large Language Model) — Большая языковая модель
- **API** (Application Programming Interface) — Интерфейс программирования приложений
- **REST** — Representational State Transfer, архитектурный стиль для веб-сервисов
- **WebSocket** — Протокол для двусторонней коммуникации в реальном времени
- **CORS** — Cross-Origin Resource Sharing, механизм для кросс-доменных запросов
- **JWT** — JSON Web Token, стандарт для передачи информации безопасно
- **CI/CD** — Continuous Integration/Continuous Deployment, практики автоматизации
- **Docker** — Платформа для контейнеризации приложений
- **PostgreSQL** — Реляционная база данных
- **Redis** — In-memory хранилище данных для кеширования
- **WSGI** — Web Server Gateway Interface, интерфейс для Python веб-приложений

12. Контрольный список развёртывания

Перед development развёртыванием:

\checkmark Клонирована ветка development
\checkmark Установлены все зависимости
\checkmark Созданы файлы .env с переменными окружения
\checkmark Инициализирована база данных
\checkmark Запущены миграции БД
\checkmark Проверена подключаемость к API провайдерам LLM
\checkmark Запущены unit тесты (успешно пройдены)
\checkmark Запущено приложение без ошибок
\checkmark Доступно на localhost:8000 (backend) и localhost:3000 (frontend)

Перед production развёртыванием:

\checkmark Все тесты пройдены (unit, integration, e2e)
\checkmark Code review завершён и одобрен
\checkmark Настроена HTTPS и SSL сертификаты
\checkmark Настроены CORS политики
\checkmark Определены и настроены все переменные окружения
\checkmark Резервное копирование БД включено и протестировано
\checkmark Настроено логирование и мониторинг
\checkmark Настроены алерты в системе мониторинга
\checkmark Создана документация по процедурам откатов
\checkmark Проведена нагрузочное тестирование
\checkmark Security audit завершён и проблемы исправлены
\checkmark Правил backup и disaster recovery задокументированы
\checkmark Команда обучена поддерживать систему

13. Версионирование документации

Версия	Дата	Описание изменений
1.0	2025-12-11	Базовая версия документации
1.1	2025-12-11	Расширенная версия с деталями архитектуры, API, развёртывания и тестирования

Документация создана: Декабрь 2025
Версия документации: 1.1
Статус: Актуальная версия
Последнее обновление: 2025-12-11T20:50:00Z