Описание технической архитектуры программного обеспечения

«LazyDoc: Программный комплекс для заполнения медицинской документации с использованием функций искусственного интеллекта»

1. Общая характеристика архитектуры

Программный комплекс LazyDoc реализован по клиент-серверной модели и включает три основных программных контура: мобильное приложение, веб-приложение и серверную часть. Серверная часть выступает центральным узлом обработки запросов, хранения данных, интеграции с внешними сервисами и организации обмена сообщениями в режиме, близком к реальному времени.

Архитектура серверной части соответствует модульному backend-приложению на FastAPI. Логика разделена на предметные подсистемы: аутентификация, чаты, сообщения, шаблоны и протоколы, статистика, организации и специальности, интеграции с медицинскими системами, подписки и обратная связь по AI-генерации.

2. Архитектурные принципы

- единый серверный контур как точка консолидации бизнес-логики, данных и интеграций;
- разделение пользовательских интерфейсов на мобильный и веб-клиенты с обращением к общему API;
- поддержка синхронного и асинхронного обмена данными: REST/JSON, SSE и WebSocket;
- выделение специализированных интеграционных адаптеров для ASR, LLM, МИС, SMTP и платёжного шлюза;
- использование шаблонов и пользовательских протоколов как отдельного слоя представления медицинской документации;
- расширяемость брендированных клиентских сборок без изменения базовой серверной логики.

3. Логическая схема архитектурных уровней

Уровень	Назначение	Компоненты
Пользовательский доступ	Точки входа пользователей системы	Врач/пользователь, браузер, мобильное устройство Android/iOS
Клиентский уровень	Интерфейсы взаимодействия и локальная обработка	Flutter mobile app; React/TypeScript web app; локальное кэширование; работа с аудио и формами
Прикладной уровень	Бизнес-логика, авторизация, генерация и обмен сообщениями	FastAPI backend: auth, chats, messages, protocols, statistics, subscriptions, organizations, MIS, external_api, ai_feedback
Уровень данных	Постоянное хранение прикладных данных и	SQLite по умолчанию через SQLAlchemy; файловая система для

	файлов	аватаров и загруженных аудиофайлов; миграции Alembic
Интеграционный уровень	Внешние сервисы и корпоративные системы	OpenAI API, Qwen ASR, GigaAM, GPTDB API, SMTP, внешние МИС/справочники специальностей

4. Клиентский контур

4.1. Мобильное приложение

Мобильный клиент реализован на Flutter/Dart. Выделены feature-модули для аутентификации, чатов, шаблонов/протоколов, статистики, настроек, подписок, интеграции с МИС и административных функций. Навигация построена через AutoRoute, управление состоянием – через flutter_bloc, сетевое взаимодействие – через Dio.

Технические особенности: локальное кэширование и сохранение данных в SharedPreferences и Hive; поддержка записи и отправки аудио; WebSocket-подключение для сообщений в чате; механизмы офлайн-очереди и последующей синхронизации аудиосообщений; использование WebView для платёжных сценариев; мультиязычная и тематическая настройка интерфейса.

4.2. Веб-приложение

Веб-клиент реализован на React/TypeScript с использованием react-scripts, React Router, Redux Toolkit и Axios. Клиент поддерживает брендирование сборок, а также генерацию брендированных manifest-файлов.

Веб-приложение обеспечивает регистрацию и вход пользователей, работу с чатами, создание и редактирование шаблонов протоколов, просмотр статистики, административные представления и специализированные сценарии интеграции.

4.3. Каналы взаимодействия клиентов с сервером

Параметр	Описание
Основной прикладной интерфейс	REST API по HTTP(S) с передачей JSON-структур.
Реальное время	SSE-потоки для событий по чатам и глобальным обновлениям, WebSocket для двустороннего обмена событиями и сообщениями.
Передача файлов	Multipart-запросы для передачи аудиофайлов и документов, используемых при распознавании и генерации протоколов.
Авторизация	JWT access/refresh tokens с проверкой прав доступа на серверной стороне.

5. Серверный контур

Серверная часть реализована на Python/FastAPI и выполняет роль центрального orchestration-слоя для всех клиентских приложений и внешних интеграций.

5.1. Основные backend-подсистемы

Подсистема	Назначение
auth	Регистрация, вход, JWT-токены, refresh-токены, коды подтверждения, профили пользователей и аватары.
chats	Управление чатами, привязка чатов к пользователям и специальностям.
messages	Хранение текстовых, JSON- и аудиосообщений; загрузка аудиофайлов; фоновое распознавание речи.
protocols	Работа с пользовательскими шаблонами протоколов, генерация протоколов из чатов и документов, экспорт и преобразование структур.
statistics	Получение агрегированной и детализированной статистики по пользователям, организациям и специальностям.
subscriptions	Тарифы, подписки, лимиты протоколов, платежи и сверка статусов с платёжным шлюзом.
organizations / specialties	Справочники организаций и специальностей, привязка пользователей и чатов к организационной структуре.
mis	Интеграция с медицинскими информационными системами и корпоративными внешними API.
external_api	Проксирование и адаптация запросов к внешнему GPTDB API и связанным сервисам.
sse / websocket	Механизмы доставки событий и сообщений в реальном времени.
ai_feedback	Хранение и обработка обратной связи по результатам работы AI-функций.

5.2. Прикладная модель данных

В серверной части выделены сущности пользователей, ролей, организаций, специальностей, чатов, сообщений, протоколов, пользовательских шаблонов протоколов, статистики, подписок и платежей. Для безопасности сессий используются отдельные сущности refresh-токенов и кодов подтверждения. Сообщения поддерживают несколько форматов: текст, JSON и аудио, что позволяет объединить диалоговый сценарий, структурированный результат AI-обработки и исходный голосовой ввод.

5.3. Техническая организация серверного приложения

- framework: FastAPI;
- ORM и доступ к данным: SQLAlchemy, aiosqlite, sync/async session makers;
- миграции схемы данных: Alembic;
- сервер запуска: Uvicorn; production-process manager: PM2;
- статические каталоги: /static и /avatars;
- фоновые задачи жизненного цикла: очистка refresh-токенов, очистка AI-feedback, сверка pending-платежей с Т-Банком.

6. Подсистема хранения данных

Серверная часть использует базу данных SQLite, подключаемую через URL формата sqlite+aiosqlite. Архитектурно доступ к данным инкапсулирован через SQLAlchemy и DAO-слой, что допускает перенос на иную СУБД при соответствующей настройке окружения.

Помимо реляционного хранения прикладных сущностей, система использует файловую подсистему для хранения пользовательских аватаров и загружаемых аудиофайлов. В мобильном клиенте дополнительно используется локальное хранилище Hive для кеширования сообщений и офлайн-аудио, а SharedPreferences – для прикладных пользовательских настроек и токенов/состояния приложения.

Параметр	Описание
Транзакционные данные	Пользователи, роли, организации, специальности, чаты, сообщения, протоколы, подписки, платежи, статистика.
Локальные файлы сервера	Аватары пользователей, загруженные аудиофайлы, статические ресурсы.
Локальные данные мобильного клиента	Hive-кеш чатов и аудио, SharedPreferences.
Схема данных	Версионируется миграциями Alembic.

7. Интеграционный слой

Архитектура системы предполагает активное взаимодействие с внешними прикладными сервисами:

- AI и LLM-сервисы: OpenAI API и внешний GPTDB API для генерации и структурирования медицинских протоколов;
- ASR-провайдеры: Qwen ASR и GigaAM для транскрибации аудиофайлов;
- медицинские информационные системы: внешние API МИС и специализированные endpoints для обмена данными;
- платёжная инфраструктура: шлюз банка для обработки подписок и сверки статусов платежей;
- почтовая инфраструктура: SMTP для служебных уведомлений и сценариев подтверждения/оповещения.

Интеграционный слой реализован не на стороне клиентов, а в основном в backend-компоненте, что позволяет централизовать авторизацию, контроль доступа, аудит сценариев и трансформацию данных между внутренними структурами системы и внешними форматами.

8. Основные прикладные потоки данных

8.1. Голосовой ввод и распознавание речи

Пользовательский клиент записывает аудио и передает его в backend в виде multipart-запроса. Сервер сохраняет исходный аудиофайл, создаёт объект сообщения типа AUDIO и инициирует фоновую задачу транскрибации. В зависимости от конфигурации выбирается провайдер ASR. После завершения распознавания текст добавляется в сообщение и доставляется клиентам по SSE/WebSocket-каналам.

8.2. Генерация протокола из чата

Для выбранного чата backend извлекает историю сообщений, при необходимости дополняет запрос пользовательским шаблоном протокола, обращается к внешнему AI/API-сервису и формирует структурированный результат. На время выполнения создаётся промежуточное сервисное сообщение, а итог доставляется в интерфейсы пользователей в асинхронном режиме.

8.3. Генерация протокола из документа

Сервер принимает PDF, Word или Excel документ, извлекает текст стандартными парсерами, а для PDF при необходимости может использовать OCR. Затем извлечённый текст анализируется AI-модулем, результат преобразуется в JSON-структуру протокола и возвращается клиенту с оценкой confidence и временем обработки.

8.4. Платные функции и подписочная модель

Логика подписок контролируется сервером: перед выполнением лимитируемых сценариев проверяется доступная квота, платежи и статус подписки. Отдельный фоновый процесс периодически сверяет неопределённые статусы платежей с банковским шлюзом.

9. Безопасность архитектуры

Используются следующие базовые механизмы безопасности:

- JWT access/refresh модель с хранением refresh-токенов и возможностью их отзыва;
- ролевой доступ и проверка принадлежности ресурсов пользователю;
- security headers middleware: CSP, HSTS для HTTPS, X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Permissions-Policy;
- контролируемая CORS-конфигурация;
- отдельные механизмы очистки устаревших refresh-токенов и обработка ошибок валидации без утечки токенов в логах;
- возможность хранения и обработки данных в изолированном серверном контуре, без прямого подключения клиентов к внешним медицинским и AI-системам.

Следует учитывать, что требования по защите персональных данных и медицинской тайны определяются не только исходным кодом, но и фактической инфраструктурой развертывания, настройкой сетевого периметра, журналированием, резервным копированием и организационными мерами оператора.

10. Эксплуатационная архитектура и развертывание

Типовой эксплуатационный контур: backend разворачивается как Python-приложение в виртуальном окружении, миграции базы применяются через Alembic, запуск осуществляется через Uvicorn, а процесс в production может управляться через PM2. Веб-приложение собирается через npm/react-scripts в статический bundle, мобильное приложение – стандартными средствами Flutter для Android и iOS.

Параметр	Описание
Backend runtime	Python 3.x, virtualenv, Uvicorn, PM2.
Web build/runtime	Node.js/npm на стадии сборки, статическая публикация web-bundle.
Mobile build/runtime	Flutter SDK, Android/iOS toolchains.
Контейнеризация	В текущих проанализированных репозиториях конфигурация контейнеризации не подтверждена.
База данных по умолчанию	SQLite; схема версии через Alembic.