Сопроводительная документация

к решению по задаче №9

«Сервис извлечения и индексирования информации из образов архивных документов (Ретроконверсия)»

Конкурс «Лидеры Цифровой Трансформации 2025»

Команда №12 «MISIS_MSc»

Содержание

1 Описание предметной области..... 3

2 Требования к сервису и их реализация.....4

2.1 Веб-сервис.....4

2.2 Интеллектуальная система распознавания..... 7

3 Общая архитектура решения.....9

4 Технологический стек.....12

5 Инструкция по развертыванию..... 14

5.1 Локальный запуск хоста..... 14

5.2 Минимальные характеристики..... 14

6 Ограничения и дальнейшее развитие.....16

Источники.....17

1 Описание предметной области

Архивные учреждения работают с массивами оцифрованных документов (образы дел, фондов, описей), значительная часть которых представлена метрическими книгами - реестрами актов гражданского состояния за XVIII-1917 гг. Документы часто гибридные (печатный + рукописный текст), содержат дореволюционную и современную орфографию и требуют трудоемкой ручной расшифровки и поиска по архивным шифрам (Ф.О.Д). При растущем притоке бумажных дел, дефиците специалистов по дореволюционной графике и повышенном общественном спросе на генеалогические/исторические исследования ручные процедуры становятся узким местом для своевременного исполнения запросов граждан.

Назначение сервиса - автоматизировать ретроконверсию: извлекать данные из образов, индексировать их и предоставлять средства верификации, тем самым сокращая трудозатраты и ускоряя наполнение научно-справочного аппарата архивов.

Функциональная сущность включает: предобработку изображений (нормализация, коррекция и т. п.), интеллектуальное распознавание печатного и рукописного текста, атрибутивное извлечение (ФИО, даты, адреса, архивные шифры) с привязкой к координатам на изображении, пользовательскую проверку/коррекцию и экспорт результатов по заданным критериям.

Ожидаемый эффект - снижение ручных операций при поиске и описании, ускорение ответа на архивные запросы и повышение доступности исторической информации.

2 Требования к сервису и их реализация

Ниже зафиксированы и описаны заявленные требования, а также принятые решения по их реализации. Статус выполнения заполнен по итогам финального этапа проекта.

2.1 Веб-сервис

Функциональные требования, предъявляемые к веб-сервису представлены в таблице 1.

Таблица 1 - Требования к веб-сервису

№	Требование	Реализация	Статус выполнения
1.1.1	Отображать список загруженных в систему документов и их атрибуты (архивный шифр, название, статус, дата загрузки), их общее число, а также число документов, загруженных в течение пользовательского сеанса. Предоставить возможность сортировки и фильтрации списка по атрибутам.	- индексированные поля в БД; - серверная пагинация; - UI-таблица с контролами сортировки/фильтров; - агрегация COUNT(*) и кэширование запросов;	Выполнено

№	Требование	Реализация	Статус выполнения
1.1.2	Для выбранного документа отображать его параметры, а также список образов в нем с предпросмотром изображения.	- хранение оригинального изображения и миниатюры; - lazy load список превью образов;	Выполнено
1.2	Искать по наименованию документа среди общего списка документов.	- параметр query в API; - debounce в 500 мс на клиенте;	Выполнено
1.3	Загружать файлы/директорию (форматы JPG/JPEG, TIFF) в рамках одного архивного документа для последующей обработки моделью.	- множественная загрузка + drag&drop; - выбор режима загрузки (файлы/директория); - ручной ввод архивного шифра (для корректной идентификации документа); - серверная нормализация входа; - валидация форматов; - присвоение общего id;	Выполнено
1.4	Отображать статус обработки документа.	- статусы: processing, success, error;	Выполнено
1.5	Удалять документы из системы.	- полное удаление данных документа из базы; - функция множественного удаления;	Выполнено
1.6	Настраивать пороги уверенности (верхний/нижний) в рамках обработки одного документа.	- хранение confidence_low/high в параметрах документа; - в зависимости от процента уверенности блока (атрибут или текст) задается уровень уверенности (низкий, средний, высокий) исходя из нижнего и верхнего порогов;	Выполнено

№	Требование	Реализация	Статус выполнения
		- применение при подсветке текста (для атрибутов - после реализации п. 2.3);	
1.7	Скачивать документы.	- выдача исходных файлов изображений образов документа через zip-архив;	Выполнено частично
1.8	Постраничный показ для каждого образа: отображать распознанный текст, атрибуты, уверенность в распознавании и статистику.	- двухпанельный интерфейс (изображение + текст/атрибуты) с возможностью пользовательской настройки ширины панелей; - масштабируемое окно просмотра изображения; - цветовая индикация уровней уверенности блоков; - навигация по образам документа;	Выполнено за исключением атрибутивного распознавания
1.9	Переключать режимы «Просмотр» и «Редактирование».	- переключатель режима с изменением UI для каждого блока;	Выполнено
1.10	Редактировать распознанные поля (текст образа и его атрибуты) и сохранять их в системе.	- редактирование полей с возможностью полного удаления; - индикация отредактированных пользователем полей;	Выполнено
1.11	Просматривать положение блоков на образе.	- выделение атрибута / текстового блока инициирует подсветку соответствующего фрагмента на изображении (поддержка подсветки атрибутов будет доступна после реализации п. 2.3); - цвет подсветки зависит от уровня уверенности блока;	Выполнено

№	Требование	Реализация	Статус выполнения
1.12	Выполнять поиск по подстроке в оцифрованном тексте.	- поиск совпадающих с подстрокой текстовых блоков в рамках выбранного образа;	Выполнено
1.13	Экспортировать результаты распознавания по параметрам, заданным пользователем.	- профили экспорта (CSV/XLSX/XML); - выбор режима экспорта (сквозное распознавание / атрибутивное распознавание); - выбор списка атрибутов для экспорта; - настройка полей экспорта (включать/не включать архивный шифр, уверенность); - возможность сохранения исходного изображения образа;	Выполнено для сквозного распознавания

2.2 Интеллектуальная система распознавания

Требования, предъявляемые к системе распознавания представлены в таблице 2.

Таблица 2 - Требования к модели

№	Требование	Реализация	Статус выполнения
2.1	Предварительно обрабатывать изображения перед подачей в систему распознавания.	- уменьшение шума изображения с помощью Гауссовского размытия; - адаптивная бинаризация изображения; - масштабирование изображения;	Выполнено
2.2	Распознавать рукописный и печатный текста с показателем WER ≤ 5%, рассчитывать уверенности по каждому образу, извлекать bbox слов.	- обучение модели OCR с возможностью детекции строк рукописного текста и их распознавания на дореволюционных текстах;	Выполнено с WER > 15% Возможно повышение качества за счет дообучения
2.3	Распознавать заданные атрибуты (ФИО, даты, адреса, архивные шифры).	- NER на русском дореформенном/современном корпусе (BiLSTM-CRF/Transformer); - привязка сущностей к bbox через выравнивание токенов;	Не выполнено в связи с ограничениями по потребляемым ресурсам
2.4	Дообучать на новых данных и правках верификатора.	- контур активного обучения: накопление размеченных правок, периодический fine-tuning с контрольными наборами; - версионирование моделей, откат по метрикам;	Не выполнено в связи с ограничениями по потребляемым ресурсам
2.5	Работать в закрытом контуре	- оффлайн-развертывание, локальные модели; - отсутствие внешних API;	Выполнено

3 Общая архитектура решения

Решение развертывается в изолированной внутренней сети Docker и состоит из следующих контейнеров: NGINX (reverse-proxy), Backend (Golang), хранилище объектов S3, PostgreSQL, Apache Kafka, ML-сервис (рисунок 1).

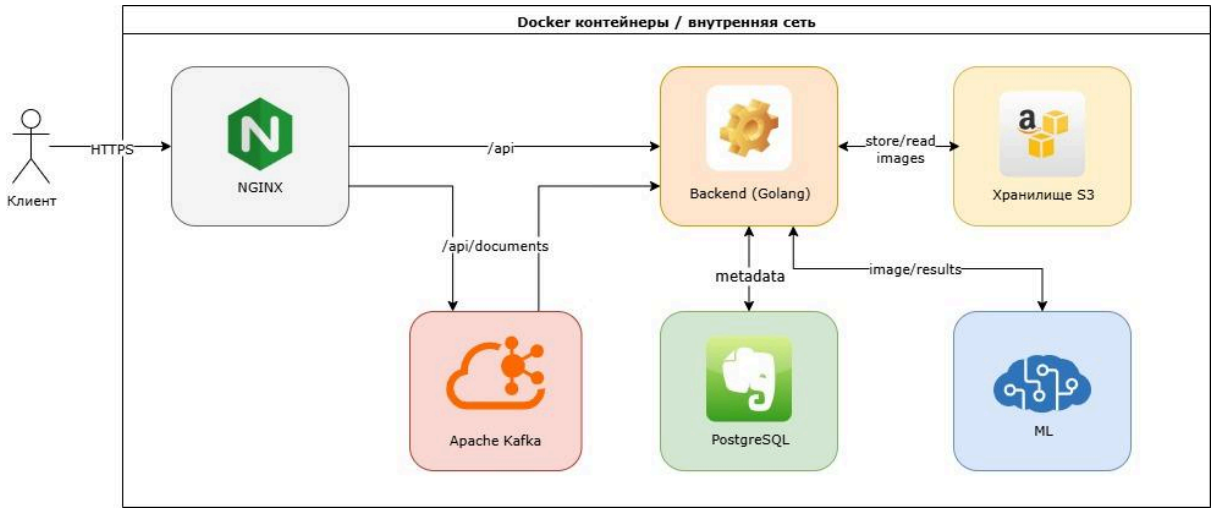

Рисунок 1 - Архитектура решения

1) NGINX:

— завершает TLS, маршрутизирует внешние запросы клиента на /api в контейнер Backend;

— ограничивает список доступных методов, внедряет заголовки безопасности и лимиты на размер тел запросов при загрузке файлов.

2) Backend (Golang):

— единая точка входа бизнес-логики;

— принимает загрузки (файлы/папки), отправляет их в очередь сообщений, валидирует форматы, создает запись документа;

— сохраняет данные в S3, метаданные - в PostgreSQL;

— инициирует обработку в ML-сервисе, передавая ссылку на образ в S3 и идентификатор страницы;
— принимает результаты распознавания (текст, атрибуты, confidence, координаты bbox), сохраняет в PostgreSQL;
— предоставляет REST API для списка документов, страниц, статусов и т.д.

3) Apache Kafka:
— очередь сообщений для обработки загруженных документов.

4) S3-хранилище:
— хранит исходные изображения;
— Backend выполняет операции store/read, ML-сервис читает образы.

5) PostgreSQL:
— хранит реляционные метаданные: документы, образы, статусы обработки, ссылки на объекты в S3, распознанные сущности, координаты, уровни уверенности.

6) ML-сервис:
— получает от Backend идентификатор страницы и подписанную HTTP-ссылку на объект в S3, выполняет предобработку, распознавание текста (и атрибутов после реализации п. 2.3), формирует координаты и confidence и возвращает результаты в Backend;
— интерфейс взаимодействия - HTTP-эндпоинты.

Потоки данных (основные сценарии):

— *Загрузка*;

Клиент → HTTPS → NGINX → Backend → (объекты) S3 → Kafka; Backend записывает метаданные в PostgreSQL, S3, отсылает сообщение в Kafka и отдает ответ пользователю, затем по очереди читает сообщения из Kafka и запускает обработку в ML.

— *Обработка*;

ML-сервис получает образ по подписанной HTTP-ссылке на объект в S3 → вычисляет результаты распознавания → результаты передаются обратно и сохраняются в БД/объектное хранилище.

— *Просмотр/редактирование*;

Клиент запрашивает /api → Backend читает из PostgreSQL и формирует подписанные ссылки на S3 для превью/страниц; правки пользователя сохраняются в PostgreSQL.

— *Экспорт*;

Backend агрегирует данные из PostgreSQL, формирует выгрузку (CSV/XLSX/XML), публикует файл в S3 и возвращает ссылку на скачивание.

4 Технологический стек

Разработка выполнена на связке веб-клиента и серверного API, развернутых в Docker и соединенных внутренней сетью за обратным прокси NGINX. Взаимодействие с ML-сервисом осуществляется по HTTP во внутренней сети Docker. Компоненты конфигурируются через переменные окружения, файл .env и health-checks.

Клиентская часть - одностраничное приложение на React TypeScript с модульной структурой и стилями SCSS. Интерфейс обеспечивает загрузку файлов, работу со списками документов и страниц, постраничный просмотр изображений с наложением координат блоков (bbox), режимы «Просмотр/Редактирование», поиск и экспорт. Обмен с сервером осуществляется по REST. Сборка проекта осуществляется посредством react-scripts.

Серверная часть реализована на Go и инкапсулирует бизнес-логику: прием и валидацию загрузок, постановку задач в Kafka, оркестрацию обработки, агрегирование результатов распознавания и формирование выгрузок. Бинарные объекты (исходные образы, превью и артефакты) хранятся в S3-совместимом объектном хранилище (MinIO), а реляционные метаданные - в PostgreSQL. Сервер предоставляет подписанные ссылки для безопасной выдачи больших файлов. Задания выполняются асинхронно, статусы и метрики обработки фиксируются для последующего аудита.

ML-контур представляет собой Python-обертку (FastAPI) над конвейером предобработки и распознавания: OpenCV применяет выравнивание, нормализацию контраста и бинаризацию, PaddleOCR выполняет распознавание печатного и рукописного текста с расчетом confidence и извлечением координат слов, NumPy обеспечивает эффективные численные операции. Результаты возвращаются серверу для записи в PostgreSQL.

В качестве основы для задачи распознавания текста использовалась модель PP-OCRv5_server_rec от PaddlePaddle с возможностью детекции и распознавания, подходящая как для рукописного, так и для печатного текстов [1]. Модель обучена для мультиязычных корпусов и обеспечивает извлечение строк, их распознавание и расчет метрик качества. Далее была организована ручная разметка дореволюционных

метрических книг из набора данных, предоставленного организаторами: на страницах формировались построчные фрагменты и соответствующие им эталонные расшифровки, которые в последующем были предобработаны (шумоподавление, бинаризация, масштабирование).

Для снижения количества ошибок в результирующем тексте введен ограничивающий алфавит допустимых символов. Дообучение выполнялось на GPU в течение 30 эпох. Лучшая версия на отложенной выборке показала точность (accuracy) = 0.60, а метрика близости строк достигла значения 0.85. Так как большинство ошибок связаны с неправильным распознаванием букв схожих по начертанию, например, «к», «и», «н», WER остался более 15%. Данный показатель возможно улучшить за счет обучения на большем количестве примеров размеченных рукописей.

5 Инструкция по развертыванию

Предусловия:

— Linux x86_64 с установленными Docker Engine и Docker Compose v2.
— Открыты порты, указанные в docker-compose-nossl.yaml.
— Достаточно свободного места на диске для образов Docker и загружаемых изображений.

5.1 Локальный запуск хоста

1) *Клонирование репозитория:*

— git clone https://github.com/kirenate/lct
— cd lct/pkg

2) *Загрузка весов модели:* скачайте веса модели и поместите их в каталог lct/pkg/.data (на хосте). В контейнере ML они будут доступны по пути /.data.

3) *Подготовка переменных окружения:*

Проверьте и при необходимости отредактируйте файл pkg/backend/.env (он монтируется в контейнер backend). Значения должны соответствовать именам переменных, которые использует сервис. Если файла нет - создайте его по этому пути.

4) *Локальный запуск контейнеров:*

— docker compose -f docker-compose-nossl.yaml pull
— docker compose -f docker-compose-nossl.yaml up -d --build

После успешного старта приложение доступно по адресу сервера по HTTP.

5.2 Минимальные характеристики

Минимальные характеристики для запуска хоста прототипа включают:

— *CPU:* x86_64, ≥ 4 vCPU;
— *RAM:* ≥ 8 ГБ;

— *Диск:* ≥ 40–60 ГБ свободного SSD-пространства;

— *ПО:* Linux + Docker Engine + Docker Compose v2.

Рекомендуемая конфигурация для комфортной работы с крупными партиями изображений:

— *CPU:* 8 vCPU;

— *RAM:* 16 ГБ;

— *Диск:* ≥ 100 ГБ SSD.

Дообучение модели (п. 2.4) выполнялось на GPU, однако для запуска прототипа и инференса в текущей конфигурации достаточно CPU.

6 Ограничения и дальнейшее развитие

На текущем этапе решение представлено как рабочий прототип, ориентированный на загрузку, базовую предобработку и распознавание текста с последующим просмотром результатов. На текущий момент следующие из функций, ранее заявленных в требованиях, реализованы частично или не реализованы в рамках проекта: скачивание zip-архивов документов; атрибутивное распознавание; дообучение модели в реальном времени по правкам верификаторов; полнотекстовая индексация на уровне документа. Ограничения зафиксированы в требованиях и архитектурных решениях прототипа как задачи следующих итераций разработки.

В качестве направления развития проекта предлагается повышение точности распознавания за счет расширения корпуса размеченных материалов и регулярного дообучения моделей, а также внедрение постобработки текстов с применением LLM для нормализации орфографии, исправления типовых ошибок и приведения дат/имен к целевым форматам. Кроме того, рассматривается введение ролевой модели доступа (администратор/пользователь) и ведение журнала правок распознанных текстов, а также проведение сравнительного эксперимента с альтернативными архитектурами OCR (в т.ч. иными стеком детекции и распознавания) для выбора оптимального контура под рукописные дореформенные тексты, с последующей интеграцией в текущую Docker-архитектуру.

Источники

1 PP-OCRv5_server_rec (PaddlePaddle) - https://huggingface.co/PaddlePaddle/PP-OCRv5_server_rec

2 Сопроводительная документация - https://disk.yandex.ru/d/QxLbZHbHrfA7SQ

3 Репозиторий проекта (исходные файлы для развертывания) - https://github.com/kirenate/lct

4 Репозиторий Frontend-части - https://github.com/exsec-dev/istok

5 Презентационные материалы - https://disk.yandex.ru/d/_00a9oBNjWzOVw