УДК 004.9

# РЕАЛИЗАЦИЯ МЕТОДОВ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДОКУМЕНТОВ

Кизилова Э.А., студент гр. 12002541, I курс
Нестерова Т.А.. студент гр. 12002540, I курс,
Научный руководитель: Федоров В.И., к.т.н., доцент
Белгородский государственный национальный исследовательский университет, г. Белгород, Россия

Практическая реализация методов интеллектуальной обработки документов требует тщательного проектирования архитектуры программной системы. Простое встраивание Vision-Language модели непосредственно в код Odoo привело бы к критическим проблемам масштабируемости и надёжности.

Во-первых, модель olmOCR-2-7B в режиме FP16 потребляет около 15 ГБ видеопамяти GPU и занимает значительные вычислительные ресурсы при обработке каждого документа. Если разместить такую модель внутри процесса Odoo, каждый запрос на обработку документа блокировал бы обработку других HTTP-запросов к системе [1], так как веб-сервер Odoo работает в однопоточном режиме для каждого воркера.

Во-вторых, нагрузка на приёмную комиссию крайне неравномерна по времени: в пиковые дни приёмной кампании система может получать сотни документов в час, а в межсезонье активность практически отсутствует. Монолитная архитектура не позволила бы эффективно управлять ресурсами GPU в зависимости от текущей нагрузки.

В-третьих, модели машинного обучения требуют специфического программного окружения с конкретными версиями библиотек PyTorch, CUDA [2], трансформеров, которые могут конфликтовать с зависимостями самой платформы Odoo. Совмещение всего технологического стека в одном контейнере значительно усложнило бы развёртывание и сопровождение системы.

Решением перечисленных проблем стало проектирование микросервисной архитектуры с асинхронной обработкой через очередь сообщений. Система разделена на три независимых компонента: портал приёмной комиссии на базе Odoo, который отвечает за взаимодействие с пользователями и хранение данных; микросервис интеллектуальной обработки документов, содержащий Vision-Language модель и реализующий методы, описанные во втором разеделе; брокер сообщений RabbitMQ, обеспечивающий асинхронную передачу задач между компонентами. Такая архитектура позволяет независимо масштабировать вычислительные мощности для обработки документов, гарантирует отказоустойчивость системы при временной недоступности GPU-сервера и обеспечивает изоляцию критичных компонентов с точки зрения безопасности данных.

Практическая реализация системы осуществлялась на доступной автору вычислительной инфраструктуре, состоящей из двух основных компонентов: локального рабочего места для GPU-вычислений и удалённого сервера для размещения веб-компонентов системы.

Локальная вычислительная платформа включала персональный компьютер со следующими характеристиками: процессор AMD Ryzen 9 5900X (12 ядер, 24 потока), видеокарта NVIDIA GeForce RTX 5070 Ti с 12 ГБ видеопамяти GDDR7, 32 ГБ оперативной памяти DDR4. В качестве операционной среды использовалась подсистема Windows Subsystem for Linux версии 2 с дистрибутивом Ubuntu 22.04 LTS [3]. Выбор WSL2 обеспечивал полную совместимость с экосистемой инструментов машинного обучения и нативную поддержку CUDA для доступа к GPU. На этой платформе был развёрнут микросервис обработки документов с моделью olmOCR-2-7B, работающей в режиме половинной точности FP16.

Веб-компоненты системы размещались на виртуальном частном сервере VPS с конфигурацией: 4 виртуальных ядра CPU, 8 ГБ оперативной памяти, 120 ГБ SSD-хранилища. На сервере были развёрнуты платформа Odoo 15 с модулями приёмной комиссии, база данных PostgreSQL 14 и брокер сообщений RabbitMQ 3.12 [4]. Связь между локальным GPU-сервером и удалённым VPS осуществлялась через обратный SSH-туннель [5], который пробрасывал порт RabbitMQ с удалённого сервера на локальную машину. Такая конфигурация позволяла воркеру на локальном компьютере получать задачи из очереди RabbitMQ на VPS и отправлять результаты обработки обратно в Odoo через тот же туннель. Использование SSH-туннеля обеспечивало шифрование всего трафика между компонентами и избавляло от необходимости открывать публичные порты на локальной машине с GPU. На рисунке 1 представлена актуальная реализация архитектуры взаимодействия компонентов системы.

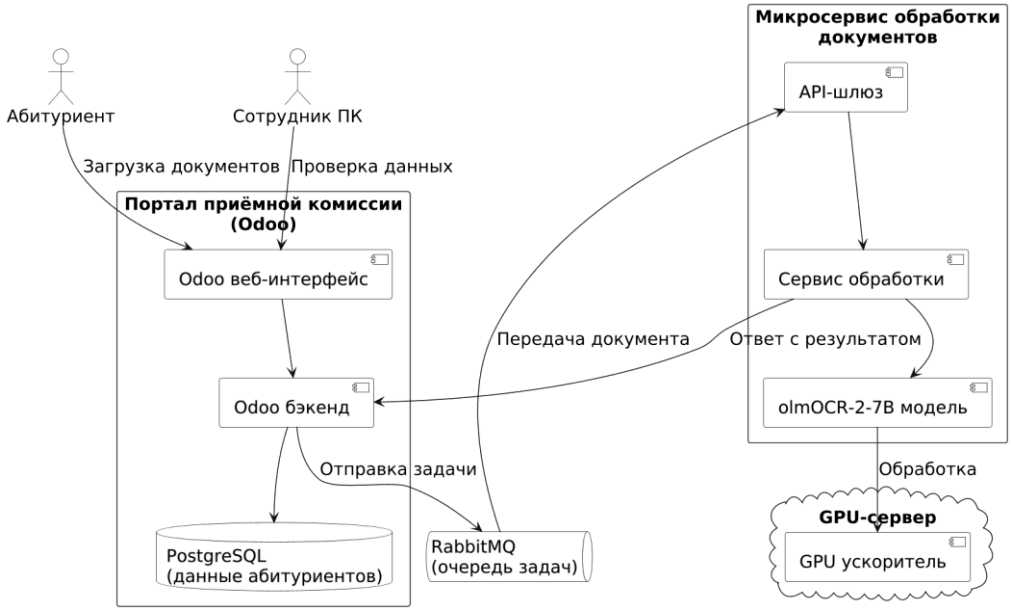


Рисунок 1 — Актуальная реализация архитектуры взаимодействия компонентов системы.

Архитектура реализованной системы построена по принципу разделения ответственности между специализированными компонентами. Odoo выполняет роль фронтенда и координатора процесса: через веб-интерфейс портала абитуриент загружает сканы документов, которые сохраняются в файловой системе сервера. После успешной загрузки Odoo формирует задачу обработки документа, присваивая ей уникальный идентификатор и фиксируя метаданные в таблице базы данных с начальным статусом «в ожидании». Сформированная задача не обрабатывается немедленно, а помещается в очередь RabbitMQ в виде JSON-сообщения, содержащего идентификатор задачи, тип документа и само изображение в формате Base64.

RabbitMQ выступает посредником между веб-приложением и вычислительным сервисом. Брокер сообщений обеспечивает надёжную доставку задач даже при временной недоступности микросервиса обработки: если GPU-сервер перезагружается или находится на обслуживании, задачи накапливаются в очереди и автоматически обрабатываются после восстановления работы сервиса. Это критически важно для стабильности работы портала в условиях пиковых нагрузок приёмной кампании. Кроме того, RabbitMQ позволяет гибко управлять приоритетами обработки: например, можно выделить отдельную очередь для срочных документов, требующих приоритетной обработки сотрудниками приёмной комиссии.

Микросервис обработки документов развёрнут на отдельном сервере с GPU и функционирует полностью автономно. Воркер-процесс подключается к RabbitMQ и ожидает поступления задач из очереди. При получении новой задачи воркер декодирует изображение из Base64, применяет методы предобработки и ориентации, загружает изображение в модель olmOCR-2 и формирует структурированный результат. Для композитных документов, таких как российский паспорт, применяется сегментированная обработка: отдельно извлекаются серия и номер с повёрнутого изображения и основные реквизиты с горизонтально ориентированного скана.

После завершения обработки микросервис не возвращает результат обратно через очередь, а отправляет его непосредственно в Odoo через HTTP-вебхук. Это решение обусловлено асимметрией размеров сообщений: задача на обработку содержит изображение размером в несколько мегабайт, тогда как результат представляет собой компактную JSON-структуру объёмом не более одного килобайта. Передача результата через вебхук позволяет избежать повторной передачи больших объёмов данных через брокер сообщений и обеспечивает мгновенное обновление статуса задачи в базе данных Odoo.

Контроллер вебхука в Odoo принимает результат обработки, верифицирует подлинность запроса по секретному ключу, переданному в HTTP-заголовке, находит соответствующую запись задачи по идентификатору и обновляет её статус на «завершена» или «ошибка» в зависимости от результата. Извлечённые структурированные данные сохраняются в поле result_data таблицы ocr.job в формате JSON. Абитуриент в своём личном

кабинете видит обновление статуса задачи в реальном времени благодаря периодическим AJAX-запросам, которые проверяют состояние всех активных задач обработки. Когда все документы успешно обработаны, интерфейс отображает форму с автоматически заполненными полями, которые абитуриент может проверить и при необходимости скорректировать перед окончательным подтверждением.

На рисунке 2 представлена архитектура асинхронной обработки документов абитуриента. Схема иллюстрирует полный цикл от момента загрузки файла через веб-интерфейс до получения структурированных данных и их отображения пользователю.

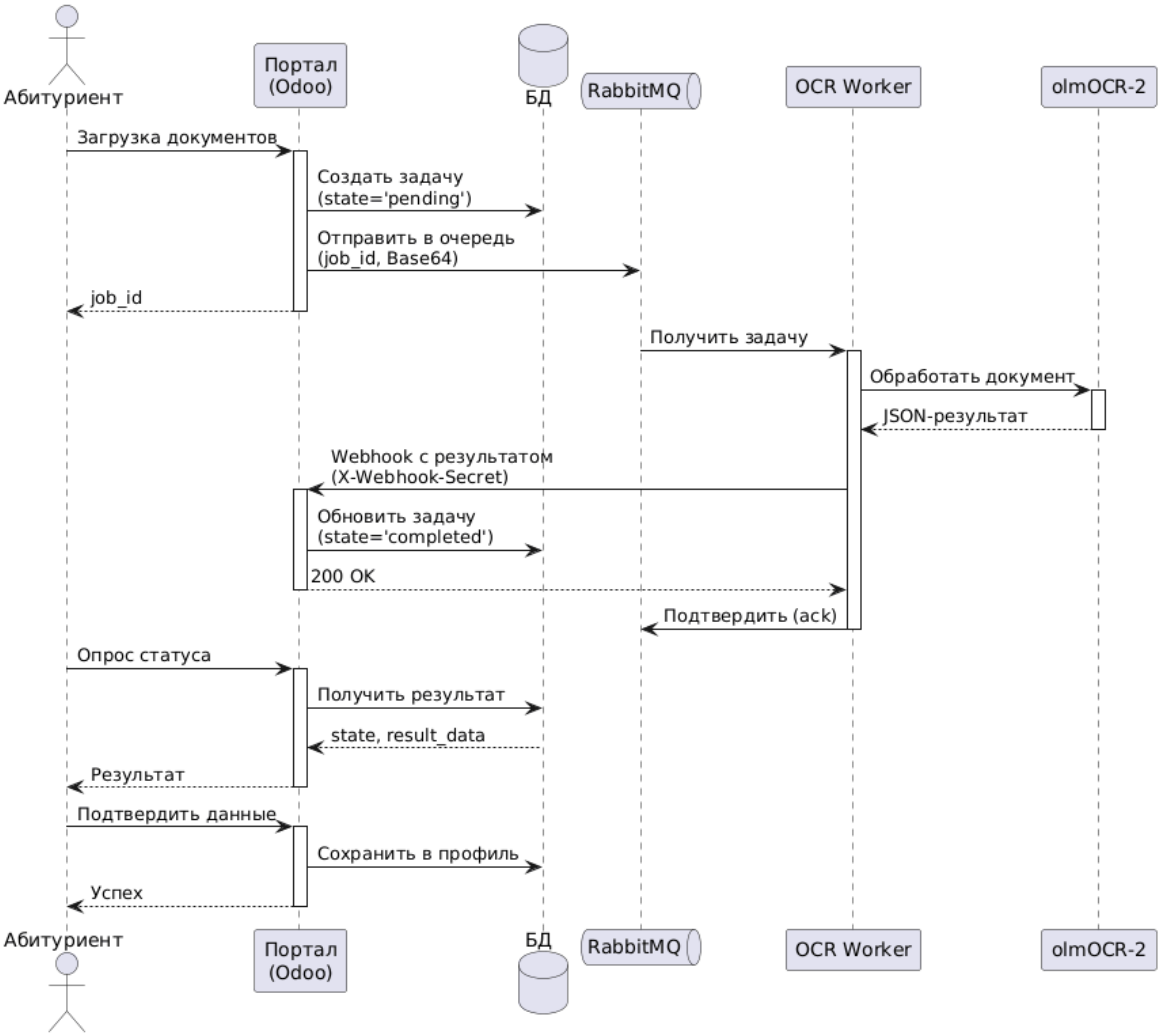


Рисунок 2 — Диаграмма последовательности взаимодействия компонентов системы при асинхронной обработке документа

Следует отметить, что обработка персональных данных абитуриентов накладывает строгие требования к архитектуре безопасности системы. Документы, загружаемые в портал приёмной комиссии, содержат паспортные данные, СНИЛС, информацию об образовании и другие сведения, относящиеся к категории персональных данных согласно Федеральному закону от 27 июля 2006 года номер 152-ФЗ [1]. Утечка или несанкционированный доступ к таким данным могут привести не только к

репутационным рискам для вуза, но и к юридической ответственности должностных лиц.

Реализованная архитектура строится на принципе защищённого периметра: все компоненты системы развёрнуты внутри закрытой сети вуза без прямого доступа из внешнего интернета. Портал Odoo доступен абитуриентам через защищённое HTTPS-соединение с актуальным SSL-сертификатом, однако непосредственное взаимодействие с базой данных PostgreSQL и микросервисом обработки документов происходит исключительно внутри внутренней сети учебного заведения. Брокер сообщений RabbitMQ также размещён в изолированном сегменте сети и недоступен извне: порты RabbitMQ открыты только для серверов Odoo и микросервиса, а аутентификация осуществляется по учётным данным с ограниченными правами доступа.

Odoo реализует многоуровневую систему контроля доступа к данным. Каждый пользователь системы принадлежит к определённой группе безопасности с чётко заданным набором прав. Абитуриенты, зарегистрированные на портале, имеют доступ исключительно к собственным данным: механизм доменов безопасности Odoo автоматически фильтрует запросы к базе данных, добавляя условие проверки владельца записи. Попытка абитуриента обратиться к задаче обработки документа другого пользователя блокируется на уровне ORM до выполнения SQL-запроса. Сотрудники приёмной комиссии получают доступ к документам абитуриентов в соответствии со своей ролью: операторы могут просматривать и проверять документы, заведующий приёмной комиссией имеет права на утверждение заявлений, а системный администратор обладает полным доступом для технического обслуживания.

Критически важным элементом безопасности является аутентификация запросов между компонентами системы. Микросервис обработки документов при отправке результата в Odoo включает в HTTP-заголовок секретный ключ, который проверяется контроллером вебхука до принятия данных. Этот ключ генерируется случайным образом при развёртывании системы, хранится в переменных окружения обоих компонентов и никогда не передаётся по незащищённым каналам. Попытка отправить результат обработки без корректного ключа приводит к отклонению запроса с кодом ответа 403 и записи предупреждения в журнал безопасности. Такой механизм защищает от подделки результатов обработки: злоумышленник, даже получив доступ к внутренней сети, не сможет внедрить ложные данные в систему без знания секретного ключа.

На рисунке 3 представлена схема сетевой топологии развёрнутой системы с указанием границ безопасности и направлений потоков данных. Схема демонстрирует разделение компонентов по сегментам сети и механизмы контроля доступа между ними [6].

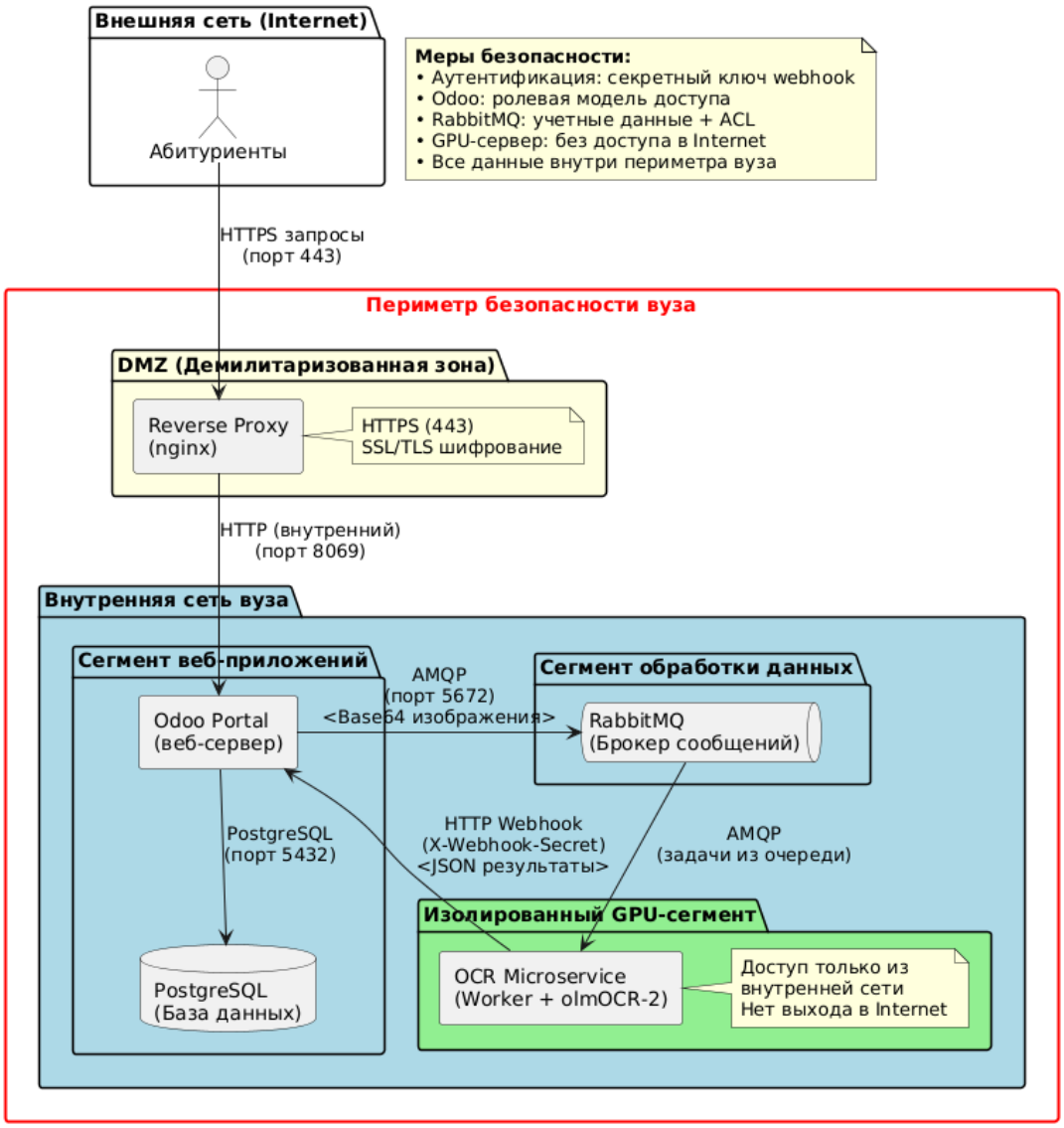


Рисунок 3 — Схема сетевой топологии системы с указанием границ безопасности и потоков данных

Реализованная модель безопасности соответствует требованиям обработки персональных данных в образовательных учреждениях. Все критические компоненты находятся под контролем вуза, исключена передача данных третьим сторонам, обеспечена защита от несанкционированного доступа на всех уровнях архитектуры. Использование открытой модели olmOCR-2 с локальным развёртыванием гарантирует, что паспортные данные и другие персональные сведения абитуриентов не покидают периметр информационной системы учебного заведения, что критически важно для соответствия законодательству о защите персональных данных.

## Список литературы:

1. Mozilla Developer Network. Web API Reference [Электронный ресурс] / Mozilla Foundation. – 2025. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API (дата обращения: 24.11.2025).

2. NVIDIA. CUDA Toolkit Documentation [Электронный ресурс] / NVIDIA Corporation. – 2025. – Режим доступа: https://docs.nvidia.com/cuda/ (дата обращения: 13.11.2025).

3. Ubuntu. Ubuntu Server Documentation [Электронный ресурс] / Canonical Ltd. – 2025. – Режим доступа: https://ubuntu.com/server/docs (дата обращения: 11.12.2025).

4. RabbitMQ. RabbitMQ Documentation [Электронный ресурс] / VMware Tanzu. – 2025. – Режим доступа: https://www.rabbitmq.com/docs/3.13/ (дата обращения: 18.12.2025).

5. SSH. Secure Shell Protocol [Электронный ресурс] / OpenSSH. – 2025. – Режим доступа: https://www.openssh.com/specs.html (дата обращения: 16.12.2025).

6. Nginx. Nginx Documentation [Электронный ресурс] / F5, Inc. – 2025. – Режим доступа: https://nginx.org/en/docs/ (дата обращения: 06.12.2025).