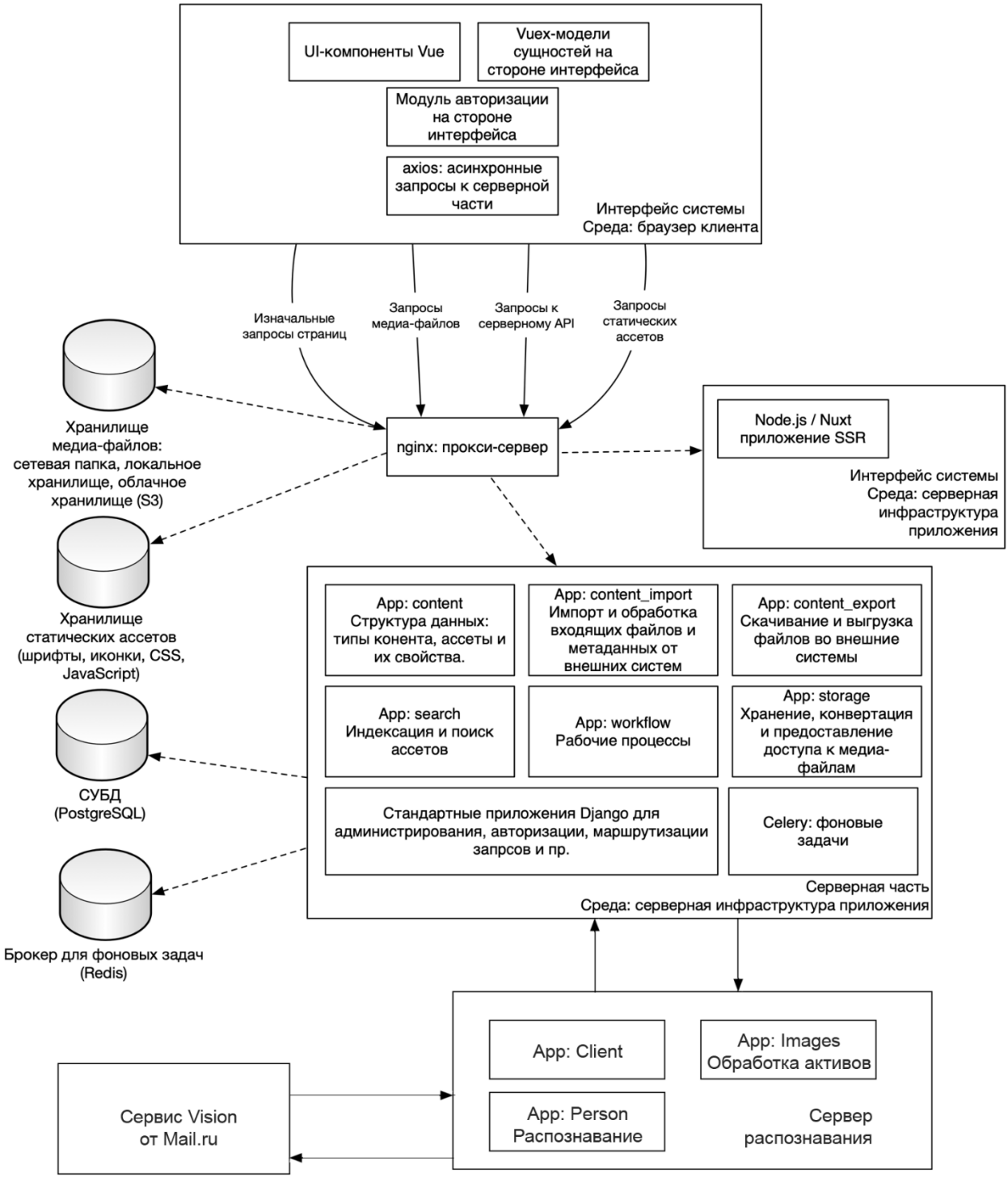
ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ «ПИКВАРИО»

Техническая архитектура системы управления цифровыми активами Picvario

Москва 2020

1. Оглавление

2. Схема


UI-компоненты Vue
Vuex-модели сущностей на стороне интерфейса
Модуль авторизации на стороне интерфейса
axios: асинхронные запросы к серверной части
Интерфейс системы
Среда: браузер клиента
Изначальные запросы страниц
Запросы медиа-файлов
Запросы к серверному API
Запросы статических ассетов
Хранилище медиа-файлов: сетевая папка, локальное хранилище, облачное хранилище (S3)
nginx: прокси-сервер
Node.js / Nuxt приложение SSR
Интерфейс системы
Среда: серверная инфраструктура приложения
Хранилище статических ассетов (шрифты, иконки, CSS, JavaScript)
App: content
Структура данных: типы конента, ассеты и их свойства.
App: content_import
Импорт и обработка входящих файлов и метаданных от внешних систем
App: content_export
Скачивание и выгрузка файлов во внешние системы
СУБД (PostgreSQL)
App: search
Индексация и поиск ассетов
App: workflow
Рабочие процессы
App: storage
Хранение, конвертация и предоставление доступа к медиа-файлам
Брокер для фоновых задач (Redis)
Стандартные приложения Django для администрирования, авторизации, маршрутизации запрсов и пр.
Celery: фоновые задачи
Серверная часть
Среда: серверная инфраструктура приложения
Сервис Vision от Mail.ru
App: Client
App: Images
Обработка активов
App: Person
Распознавание
Сервер распознавания

3. Приложение: интерфейс

Интерфейс реализуется в виде приложения, разработанного на фреймворке Nuxt 2.4+. При этом используются стандартные средства сборки фреймворка.

Конфигурация приложения для различных сред осуществляется при помощи переменных окружения.

В реализации клиентской части системы используется механизм Server-Side Rendering (SSR), являющийся штатной функцией Nuxt. Это означает, что запросы к пользовательскому интерфейсу идут к запущенному на серверной инфраструктуре сервису Node.js / Nuxt, выполняющему логику страницы и ее рендеринг.Компоненты UI

В соответствии со стандартной практикой, интерфейс разбивается на компоненты Vue.

Каждый компонент содержит разметку, оформление и логику внутри соответствующего файла компонента (.vue).

Используется БЭМ-подобная разметка для классов DOM-дерева компонентов.

Компоненты вкладываются друг в друга, образуя соответствующие экранные формы интерфейса.

Приоритетной является изоляция компонентов друг от друга, чтобы позволять их независимое друг от друга применение.

3.1. Компоненты UI

В соответствии со стандартной практикой, интерфейс системы реализован в виде набора компонентов Vue. Каждый компонент содержит разметку, оформление и логику внутри соответствующего файла компонента (.vue).

Используется БЭМ-подобная разметка для классов DOM-дерева компонентов.

Компоненты вкладываются друг в друга, образуя соответствующие экранные формы интерфейса.

Приоритетом является изоляция компонентов друг от друга, чтобы обеспечивать их независимое друг от друга применение.

3.2. Vuex

Для реализации моделей данных на стороне интерфейса используется стандартная библиотека Vuex, входящая в состав фреймворка Nuxt.

3.3. Модуль авторизации

Для авторизации используется стандартный модуль авторизации Nuxt, адаптированный для работы с django-rest-framework.

При этом используется токен авторизации, передаваемый во всех запросах к серверу. Токен авторизации имеет ограниченный срок действия.

Клиент получает токен отдельным запросом, передавая валидные логин и пароль.

Восстановление пароля осуществляется по логину.

3.4. Серверная часть системы

Функционал серверной части реализуется на фреймворке Django 2.1+.

Код серверной части распределен между логически обособленными приложениями (Django Apps).

Каждое приложение содержит модели, представления, сериализаторы API и прочий код, реализующий функции приложения и его программную интеграцию с другими приложениями, интерфейсом или внешними системами.

Программный интерфейс реализуется при помощи библиотеки django-rest-framework.

Фоновые задачи реализуются при помощи библиотеки Celery.

Функционал авторизации, маршрутизации запросов, ORM реализуются стандартными средствами фреймворка Django.

Конфигурация приложения для различных сред осуществляется при помощи переменных окружения.

3.5. Django app: admin

Приложение предоставляет административный интерфейс для суперадмина, а также возможность создания независимых клиентских пространств системы (тенантов) в рамках мультенантной архитектуры.

3.6. Django app: clients

Приложение предоставляет программный интерфейс для создания и управления тенантами.

Основная структура данных приложения — модель Client.

3.7. Django app: face_recognition

Приложение предоставляет программный интерфейс взаимодействия с сервисом распознавания: оно отправляет активы на распознавание лиц, а также принимает результат распознавания.

Основная структура данных приложения — модели Task, Person, AssetPerson.

Фоновые задачи: отправка изображений на распознавание.

3.8. Django app: jager

Приложение занимается сбором, хранением, анализом и отображением трейсов.

3.9. Django app: password_reset

Приложение по запросу отправляет email со ссылкой для восстановления пароля.

Основная структура данных — модели ResetPasswordToken.

3.10. Django app: permissions

Приложение создаёт список разрешений и управляет им.

Основная структура данных приложения — модели PublicProp, Rule.

3.11. Django app: pusher

Данное приложение реализует обмен данных по WS с интерфейсом.

3.12. Django app: uploader

Приложение реализует загрузку активов чанками.

3.13. Django app: content

Приложение реализует программный интерфейс для запроса информации об активе и его свойствах.

Основная структура данных — модели Content Type, Prop, Value, EnumValue, Asset, Prop Groups.

3.14. Django app: content_import

Приложение реализует программный интерфейс получения файла актива и его свойств для импорта.

Фоновые задачи:

- конвертация полученного файла для генерации изображений предварительного просмотра в ленте;
- создание актива в базе данных;
- логика исключения дубликатов при импорте за счет ввода внешнего идентификатора импортируемого актива.

3.15. Django app: content_export

Приложение реализует программный интерфейс запуска экспорта актива одним из доступных способов.

Фоновые задачи:

- Экспорт актива для прямой загрузки;
- Экспорт актива на один или несколько FTP-серверов.

Результат фоновой задачи, помимо статуса (успех / ошибка) может содержать URL загрузки результата (для скачивания файла).

В ходе экспорта, в файлы поддерживаемых форматов записываются метаданные.

Схема соответствия свойств актива и полей метаданных задается фиксированно.

3.16. Django app: search

Приложение реализует программный интерфейс поиска по базе активов.

Функционал приложения индексирует активы и их свойства приложения content.

Для индексации используется библиотека Haystack, совместимая с Elasticsearch и Solr.

3.17. Django app: workflow

Приложение реализует программный интерфейс запуска рабочих процессов.

Содержит рабочие процессы публикации, модификации, объединения, извлечения и отправки на распознавание.

3.18. Django app: storage

Приложение реализует иерархические модели для хранимых файлов.

Приложение содержит адаптеры для различных хранилищ: локального и объектного.

3.19. Django app: tasks

Приложение реализует программный интерфейс для получения списка фоновых задач со статусами. Содержит Модель списка фоновых задач пользователя.

4. Сервер распознавания

Функционал сервиса распознавания лиц реализуется на фреймворке Django 2.1+.

Код серверной части распределен между логически обособленными приложениями (Django Apps).

Каждое приложение содержит модели, представления, сериализаторы API и прочий код, реализующий функции приложения и его программную интеграцию с другими приложениями, интерфейсом или внешними системами.

Программный интерфейс реализуется при помощи библиотеки django-rest-framework.

Фоновые задачи реализуются при помощи библиотеки Celery.

Функционал авторизации, маршрутизации запросов, ORM реализуются стандартными средствами фреймворка Django.

Сервис конвертирует полученный актив в обрабатываемый формат и отправляет его в сторонний сервис распознавания Vision от mail.ru (mcs.mail.ru). Получает из сервиса идентификаторы персон, распознанных на отправленных активах. Сервис сохраняет идентификаторы вместе с изображениями соответствующих лиц. Сервис предоставляет возможность назначить имя персоне, а также отправляет эти имена приложению face_recognition.

4.1. Django app: client

Основная структура данных приложения — модель Client.

4.2. Django app: images

Приложение реализует функции обработки и хранения изображений.

Основная структура данных — модель Image.

4.3. Django app: person

Приложение реализует программный интерфейс, выполняющий отправку изображений в сервис Vision и обработку данных распознавания.

Основная структура данных — модель Person, ImagePerson, PersonContext.

5. Прокси-сервер

Сервер Nginx обеспечивает маршрутизацию запросов клиента между:

- Серверной частью приложения,
- Node.js / Nuxt сервисом для Server-Side Rendering,
- Хранилищами медиа-файлов и активов.

Для авторизации некоторых запросов к медиа-файлам, требующих авторизации, используется механизм X-Accel Redirect, позволяющий проверять авторизацию запроса клиента на серверной части без необходимости задействовать ее для дальнейшей передачи данных самого файла.

6. Хранилище данных

6.1. Медиа-файлы

Медиа-файлы могут быть размещены:

- В локальной файловой системе при развертывании системы на единственном сервере,
- В сетевой папке,
- В объектном хранилище, например Amazon S3.

Серверная часть приложения генерирует URL для получения конкретного файла в зависимости от его размещения:

- Доступные локально или в сетевой папке файлы передаются через прокси-сервер nginx.
- Для файлов, размещенных в объектном хранилище, генерируется соответствующий URL, по которому клиент обращается непосредственно к объектному хранилищу. При необходимости авторизации используются соответствующие механизмы этого хранилища.

Поддерживаются объектные хранилища, совместимые с библиотекой django-storages.

6.2. Статические файлы

Статические файлы генерируются в ходе сборки приложения Nuxt и доставляются клиентам прокси-сервером Nginx.

Архитектура системы позволяет в будущем использовать географически распределённую сетевую инфраструктуру (Content Delivery Network – CDN).

6.3. СУБД

В системе используется СУБД PostgreSQL, как рекомендуемая для использования с фреймворком Django.

7. Развертывание системы

Для развертывания системы используется экосистема Docker для контейнеризации сервисов приложения:

- База данных PostgreSQL (только для тестовых сред),
- Брокер Redis (только для тестовых сред),
- Фронтэнд-сервер Node.js,
- Бэкэнд-сервер Djano / gunicorn,
- Очереди Celery,
- Уведомления Pusher,
- Поисковая система Elasticsearch,
- Панель визуализации данных Kibana,
- Celery Beat для задач по расписанию,
- Прокси-сервер Nginx.

Описание правил запуска осуществляется при помощи Docker Compose.

Для передачи переменных окружения используются стандартные механизмы Docker / Docker Compose.

При развертывании в различных средах используются одни и те же образы контейнеров, которым передаются разные переменные окружения в зависимости от среды.

В продуктивной среде не допускается запуск служб базы данных и брокера в контейнерах.