МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки/специальность
09.03.01 Информатика и вычислительная техника

направленность (профиль)/специализация
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

Разработка API онлайн-микросервиса по редактированию изображений на основе импортозамещенных веб-технологий

Обучающегося 4 курса
очной формы обучения
Нюхалова Дениса Глебовича

Руководитель выпускной квалификационной работы:
кандидат педагогических наук, доцент
Государев Илья Борисович

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многие современные web-приложения спроектированы по технологии «клиент-сервер». Клиентская часть приложения работает в браузере. Она предоставляет интерфейс, при помощи которого пользователь может взаимодействовать с программным продуктом. Серверная часть приложения выполняется на удаленном компьютере или системе компьютеров. По запросу с клиентской части, она выполняет программный код и отправляет результаты своей работы обратно пользователю. Принципы взаимодействия клиентской и серверной частей определяет API (Application Programming Interface –«программный интерфейс приложения»). API – это набор функций серверной части приложения, к которому может обращаться клиентская часть того же приложения или иные программы.

Разработчики и IT компании могут на платной или безвозмездной основе предоставлять доступ к API своих приложений. Это открывает программистам широкий инструментарий для решения разного рода задач уже в своих программных продуктах. Одной из таких задач является редактирование и обработка изображений.

Различного рода изображения повсеместно распространены в интернете. На коммерческих сайтах, в приложениях и социальных сетях, ежедневно размещается огромное количество фотографий, рисунков, схем и т.д. Люди публикуют их, как для развлечения, так и для реализации своих практических целей. Для того, чтобы пользователи и разработчики могли корректно визуализировать ту или иную информацию при помощи изображений, им, зачастую, необходима обработка или редактирование. Это позволяет улучшить зрительное восприятие изображения, удалить или добавить различные графические элементы, подготовить к печати.

Существует большое количество онлайн-сервисов для обработки изображений. Именно они, чаще всего устроены по технологии «клиент-сервер». Обработка изображений – требовательный к ресурсам компьютера процесс, поэтому, большинство функций онлайн-редакторов чаще всего выполняются на более мощных удаленных серверах.

Существует очень ограниченное количество функционирующих российских приложений и API обработки изображений. Этим обуславливается **актуальность** дипломной работы.

Объект исследования – API, предназначенные для редактирования изображений. **Предметом исследования** является серверная часть сервиса по обработке изображения.

Целью работы является разработка API для онлайн-микросервиса по редактированию изображений. Этот сервис следует создавать с использованием импортозамещенных технологий и размещать на российских серверах

В соответствии с поставленной целью необходимо решить следующие **задачи**:

- Проанализировать современные подходы к написанию онлайн сервисов и API для обработки изображений;
- Изучить отечественные сервисы, предоставляющие услуги аренды виртуальных выделенных серверов;
- Определить набор функций, которые будут выполняться на серверной части сервиса;
- Разработать серверную часть приложения и API, реализующие функции по редактированию изображений;
- Разместить код серверной части приложения на виртуальном выделенном сервере.

Теоретическая значимость заключается в определении современных подходов и принципов разработки API в сфере работы с изображениями.

Практическая значимость заключается в разработке API онлайн-микросервиса редактирования изображений для отечественных приложений.

Результатом выпускной квалификационной работы является готовый к использованию онлайн-микросервис с API, работающий на российском сервере.

Дипломная работа имеет следующую **структуру**: введение, две главы, заключение, список литературы. Выпускная квалификационная работа содержит - страниц. В работе представлено - рисунков, - таблиц.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ. ПОДГОТОВКА К РАЗРАБОТКЕ

1.1. Анализ современных подходов к написанию онлайн сервисов и API для обработки изображений

API взаимодействует с приложениями посредством обмена сообщениями. Структуру сообщений определяют протоколы передачи данных. Одним из основных таких протоколов в интернете является HTTP (HyperText Transfer Protocol — «протокол передачи гипертекста») и его расширение HTTPS с поддержкой шифрования. Он используется во многих широко известных web-API[19].

HTTP – это протокол, обеспечивающий общение между клиентскими и серверными частями web-приложений. Сообщения, отправляющиеся с клиента, называются запросами, а с сервера – ответами. Клиентом можно назвать любой инструмент или устройство, действующие от лица пользователя. Чаще всего, в роли клиента выступает веб-браузер. Сервером является одна или несколько виртуальных машин или компьютеров. Он полностью или частично обрабатывает данные, полученные с запросом, и предоставляет результат обработки в ответе клиенту. Между ними могут находится программы-посредники, так называемые прокси, которые не влияют на запрос, но способны выполнять различные вспомогательные функции, которые по тем или иным причинам не может выполнить сервер, как например, кэширование, протоколирование, фильтрация и т.д. Схематично последовательность обмена сообщениями между клиентом и сервером в рамках протокола HTTP представлена на рисунке 1 [1].

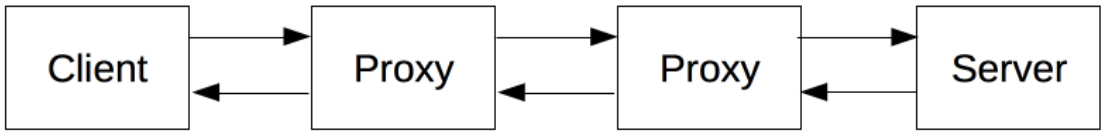

Рисунок 1 – Обмен сообщениями между клиентом и сервером, автор – MDN Web Docs

Любое сообщение в HTTP состоит из «головы» и «тела». Головная часть запроса включает в себя:

1. Стартовую строку. В ней содержится версия протокола и метод HTTP, который описывает требуемое от сервера действие. В HTTP представлено несколько методов, каждый из них может описывать то или иное действие на сервере. Методы могут отличаться на предмет ограничений по типу данных, длине запроса, видимости и т.д. Чаще всего в запросах используются методы GET и POST [2].
2. Набор заголовков. Он уточняет запрос и передает серверу дополнительную информацию. Существует несколько типов заголовков:
    - основные заголовки, содержащие информацию о самом сообщении, например дату и время запроса;
    - заголовки запроса, которые могут передавать информацию о контексте запроса, для того чтобы сервер мог адаптировать ответ;
    - заголовки сущности, передающие данные о теле запроса.

Голова запроса содержит:

1. Строку статуса. В этой строке указана версия протокола, код состояния и текст пояснения, привязанный к коду состояния. Код состояния состоит из трех цифр и указывает на результат обработки запроса [1]. Существует пять классов кодов состояния:

- информационные, в диапазоне от 100 до 199;
- успешные, в диапазоне от 200 до 299;
- перенаправления, в диапазоне от 300 до 399;
- клиентские ошибки, в диапазоне от 400 до 499;
- серверные ошибки, в диапазоне от 500 до 599;

2. Набор заголовков. Он также как и в случае с набором заголовка запроса передает дополнительную информацию, но уже об ответе. Также существует несколько типов таких заголовков:
   - основные, относящиеся также к информации о сообщении
   - заголовки ответа, передающие информацию об ответе или сервере
   - заголовки сущности, передающие данные о теле ответа

Тело HTTP сообщения – необязательная, но не менее важная часть запросов и ответов. Она отделяется от головы запроса пустой строкой и содержит данные, которые мы хотим передать в запросе. Тип данных, передаваемых в теле запроса и возвращаемых в ответе, определяется заголовком Content-Type. В этом заголовке указывается MIME-тип – идентификатор формата файлов, передаваемых в интернете[20]. Пример HTTP сообщений представлен на рисунке 2 [1].

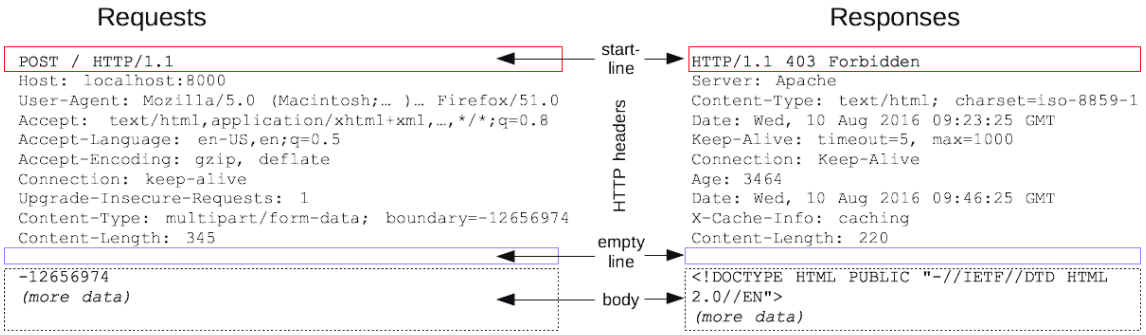

Рисунок 2 – Пример HTTP запроса и ответа, автор – MDN Web Docs

В сервисах для обработки изображений, клиент и сервер должны обмениваться HTTP сообщениями, содержащими непосредственно изображения и указания на то, как именно сервер должен его обработать.

Существует два основных подхода при обмене сообщениями в HTTP. Первый – использовать строгий стандарт написания запросов и ответов поверх HTTP – SOAP (Simple Object Access Protocol – «простой протокол доступа к объектам»). В данном случае в теле HTTP запроса и ответе сервера передается XML разметка, написанная в соответствии с языком описания WSDL (Web Services Description Language – «язык описания веб-сервисов»). Спецификации для каждого типа взаимодействия в SOAP запросах обеспечивает высокий уровень надежности и безопасности при обмене сообщениями. Второй подход – это прибегнуть к использованию архитектурного стиля REST (Representational State Transfer — «передача репрезентативного состояния») при разработке API. REST не стандартизирует тело HTTP запроса, а подразумевает под собой набор правил, позволяющих наиболее эффективно использовать возможности HTTP при обмене сообщениями [6]. Отсутствие необходимости в написании запросов определенного типа и формата обеспечивает REST относительно высокую скорость отправки запросов, а также упрощает написание RESTful приложений [5].

SOAP был одним из первых стандартов создания программируемых интерфейсов для обмена данными между системами. SOAP долгое время являлся наиболее распространенным подходом при написании API. Он остается важным стандартом для веб-сервисов и работает во многих внутренних системах по всему миру, однако для новых проектов многие организации выбирают архитектуру микросервисов с использованием REST API. REST обеспечивает более гибкий и быстрый процесс разработки приложений.

В случае клиент-серверного обмена изображениями, важную роль играет размер запросов и ответов, так как он влияет на скорость обработки HTTP

сообщений как с клиентской стороны приложения, так и с сервера. Из-за простоты применения и высокой скорости обработки запросов и ответов, большинство современных онлайн сервисов по обработке изображений построены в стиле REST. В данной дипломной работе описан процесс разработки API в соответствии с архитектурном стилем REST.

Важной частью любого онлайн редактора изображений является обработка графики. Серверная часть приложения может как выполнять узкий набор функций по обработке графики, так и проводить широкий список различных манипуляций над ней. Возможности API, зачастую, упираются в доступную для сервера мощность. Для разработки сервиса редактирования изображений было бы целесообразно использовать один многофункциональный API, так как запросы клиентской части приложения не нужно форматировать под разные серверы, что значительно упростило бы разработку и отладку.

Большинство многофункциональных API для обработки изображений не распространяются в открытом доступе. Компании, разрабатывающие онлайн-редакторы изображений не заинтересованы в том чтобы компоненты их программных продуктов использовались в сторонних проектах. Однако, некоторые из них предоставляют доступ к своим API на коммерческой основе. Одним из примеров является является сервис Pixo. В его возможности входит:

- изменение формата изображения;
- применение фильтров;
- добавление текста;
- изменение разрешения изображений;
- рисование фигур поверх изображений;
- изменение параметров яркости, контрастности, насыщенности, размытия;
- редактирование гаммы изображения;
- удаление шумов;
- редактирование заднего фона изображений [9].

API также предоставляет компания Pixelixe. Среди его возможностей можно выделить:

- сжатие изображений;
- изменение размера изображений;
- изменение разрешения изображений;
- поворот изображения на определенный угол;
- отражение изображений по вертикали и горизонтали;
- изменение параметров яркости, контрастности, насыщенности, размытия;
- применение фильтров[10].

Иные взаимодействия с изображениями, как например, рисование при помощи инструмента кисть, накладывание изображений друг на друга, выделение частей изображения и т.д. осуществляются на клиентской части онлайн-редакторов. Данные манипуляции требуют хранения на сервере состояний клиента, а это противоречит стилю архитектуры REST [6].

В рамках дипломной работы будут реализованы следующие функции:

- применение фильтров;
- поворот изображений на определенный угол;
- изменение размера изображений;
- изменение разрешения изображений;
- отражение изображений по вертикали и горизонтали;
- изменение параметров размытия;

Остальные манипуляции с изображениями будут проводиться на клиентской стороне приложения.

В результате анализа инструментов и технологий адекватных поставленной задаче была выбрана платформа Node.js и библиотека Express[14]. Среди преимуществ данной платформы выделяют:

- доступность;
- масштабируемость;
- скорость выполнения исходного кода;
- встроенный пакетный менеджер NPM [4].

Для обработки изображений использовался модуль Sharp. На рисунке 3 представлен результат сравнения скорости выполнения задачи по изменению расширения JPEG изображения со значения 2725x2225 до 720x588 и его последующим сжатием с настройкой «качества» 80 разными Node.js модулями [11].

Module	Input	Output	Ops/sec	Speed-up
jimp	buffer	buffer	0.84	1.0
squoosh-cli	file	file	1.08	1.3
squoosh-lib	buffer	buffer	1.85	2.2
mapnik	buffer	buffer	3.45	4.1
gm	buffer	buffer	8.60	10.2
gm	file	file	8.66	10.3
imagemagick	file	file	8.79	10.5
sharp	stream	stream	28.90	34.4
sharp	file	file	30.08	35.8
sharp	buffer	buffer	30.42	36.2

Рисунок 3 Сравнение скорости работы различных Node.js модулей обработки изображений

Из результатов тестирования можно сделать вывод о том, что Sharp предоставляет наиболее эффективный функционал для обработки изображений.

1.2. Анализ отечественных сервисов аренды виртуальных выделенных серверов

Для размещения web приложений, разработчики могут воспользоваться услугами компании провайдера, которая предоставляет ресурсы своих вычислительных машин для работы. Компании могут предоставить доступ как к виртуальному хостингу – дисковому пространству на своей вычислительной машине, так и к виртуальному выделенному серверу.

VDS (Virtual Dedicated Server – виртуальный выделенный сервер) имеет ряд преимуществ перед виртуальным хостингом. На виртуальном хостинге присутствует ряд ограничений. Пользователь не может устанавливать необходимое ПО, ему доступны только те инструменты, которые предоставляет компания-поставщик услуг. VDS же предоставляет полный доступ к виртуальной операционной системе с возможностью устанавливать любое ПО. На виртуальном хостинге также имеется ограничение на использование ресурсов сервера, а для VDS можно настроить необходимую под определенное приложение аппаратную конфигурацию системы [18].

VDS в выпускной квалификационной работе позволит оптимизировать ресурсы сервера под обработку изображений, а также предоставит свободу в выборе программных средств обработки.

Был проведен сравнительный анализ следующих популярных отечественных сервисов, предоставляющих аренду VPS:

- Yandex Cloud
- Reg.ru
- VK Cloud Solutions

Результат анализа представлен в таблице 1.

Таблица 1 - Сравнительный анализ популярных отечественных сервисов, предоставляющих услуги аренды VDS

Название сервиса	Наличие документации	Доступные ОС	Функциональные особенности	Стоимость
Yandex Cloud	Присутствует	Ubuntu, Windows Server, CentOS, Debian, SLES, Fedora	• Разные зоны доступности • Виртуальные машины с GPU • Поддержка интерфейсов CI систем • Управление сервером по API • Хранение данных в зашифрованном виде в соответствии с требованиями 152-ФЗ	от 741 ₽ в месяц
Reg.ru	Присутсвует	AlmaLinux, CentOS, Debian, Rocky Linux, Ubuntu	• Высокочастотные серверы 3,7+ ГГц • Шаблоны приложений • Панели ISPmanager и FASTPANE • Снэпшоты • Клонирование виртуального сервера • Приватные сети • Управление сервером по	от 370 ₽ в месяц.

			API • Остановка сервера с оплатой только за диск и IP • Резервное копирование	
VK Cloud Solutions	Присутсвует	Windows, Linux	• Безлимитный трафик 1 Gbit/s • Балансировщик нагрузки • Private & Public DNS • VPN и фильтрация трафика • Бесплатные внешние статические IP-адреса • ЦОДы Tier III в РФ. Intel® Xeon® Gold • Гарантированная доля vCPU 100%	от 1110 ₽ в месяц.

В связи с наличием различных зон доступности, наибольшего выбора ОС, хранением данных в зашифрованном виде в соответствии с требованиями 152-ФЗ и более комфортной цены по сравнению с VK Cloud Soutions, для размещения API и серверной части приложения был выбран сервис Yandex Cloud.

Выводы по главе 1:

1. Был проведен анализ различных подходов к написанию API. Принято решение об использовании архитектурного стиля REST при разработке.

2. В соответствии с поставленной задачей был определен набор функций, которые будут выполняться на серверной части онлайн-сервиса.
3. Были изучены отечественные сервисы, предоставляющие услуги аренды виртуальных выделенных серверов и сделан вывод о том, какой сервис использовать в выпускной квалификационной работе.

ГЛАВА 2. РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ГРАФИЧЕСКОГО РЕДАКТОРА И API ПО ОБРАБОТКЕ ИЗОБРАЖЕНИЙ

Процесс разработки приложения проводился в с соответствии с требованиями к разработке и документированию из ГОСТ Р 51904-2002 [7]. Для программного продукта, описываемого в выпускной квалификационной работе были выделены следующие этапы разработки:

- определение требований к ПО;
- проектирование ПО;
- кодирование ПО;
- интеграция ПО.

2.1 Определение требований к API

В рамках разработки приложения было составлено техническое задание. В соответствии с ним и выводами, сделанными в первой главе, был сформирован следующий список требований к API разрабатываемого онлайн-сервиса обработки изображений:

1. Приложение должно строиться по клиент-серверной архитектуре. Разработка серверной части приложения должна вестись в архитектурном стиле REST.

2. Взаимодействие двух частей приложения приложения ведется по протоколу HTTP.

3. Клиентская часть обращается к серверу только по одному URL-адресу и только по одному протоколу – HTTP.
4. В соответствии с архитектурным стилем REST, запрос к серверу должен содержать всю необходимую информацию для обработки изображения.

5. Запросы будут обрабатываться библиотекой Express платформы Node.js.
6. Средствами платформы Node.js, на серверной части должны производиться следующие операции над изображениями:
    - применение фильтров;
    - поворот изображений на определенный угол;
    - изменение размера изображений;
    - изменение разрешения изображений;
    - отражение изображений по вертикали и горизонтали;
    - изменение параметров размытия изображения;

2.2 Проектирование API

На этапе проектирования была составлена Use-case диаграмма приложения. Диаграмма представлена на рисунке 1 приложения А. По данной диаграмме можно сделать следующие выводы:

1. Определены функции приложения, при обращении к которым будут отправляться запросы к API.
2. Для изменения разрешения и размера изображения требуется обращения к функции определения метаданных.
3. Функции по размытию изображения, отражению изображения и применению фильтров не включают в себя вызов иных функций.

Также была составлена блок-схема, которая отражает принципы работы серверной части приложения. Она представлена на рисунке 4.

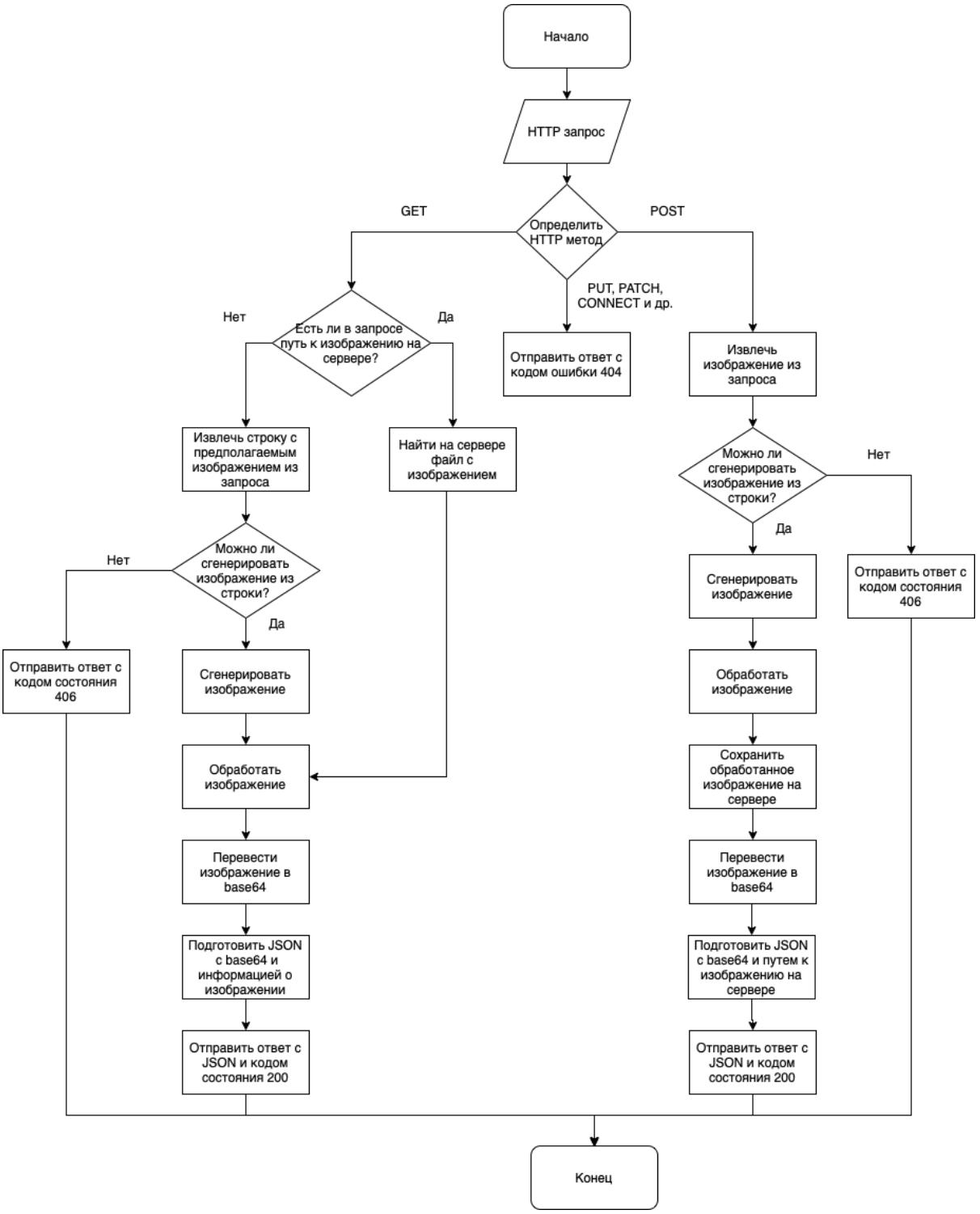

Рисунок 4 – Блок-схема алгоритма по обработке изображений с сервера.

На этапе проектирования, было принято решение о том, что изображения будут передаваться в формате base64. Это формат кодирования двоичных данных при помощи символов ASCII (American standard code for information interchange – Американский стандартный код для обмена информацией) [15].

Ответы с сервера будут приходить клиенту в формате JSON. Разработка клиентской части ведется при помощи языка программирования JavaScript. У этого языка присутствуют инструменты для быстрой обработки данных в этом формате [16]. Эти решения отражены на блок-схеме. В соответствии с ней и use-case диаграммой проходил процесс кодирования.

2.3 Кодирование API

Кодирование API осуществлялось в редакторе исходного кода Visual Studio Code. Среди его функциональных особенностей, использованных в процессе разработки можно выделить:

- интеграцию с системой контроля версий Git;
- возможность подключения к удаленному серверу по протоколу SSH
- пользовательские расширения.

Вспомогательным инструментом при разработке являлся сервис GitHub Desktop. Это графический интерфейс для управления репозиториями веб-сервиса GitHub. В данной дипломной работе GitHub также использовался для размещения и резервного копирования исходного кода клиентской и серверной частей приложения, а также для поиска дополнительных пакетов Node.js для использования в разработке.

Структура серверной части приложения, созданного в Visual Studio Code представлена на рисунке 5.

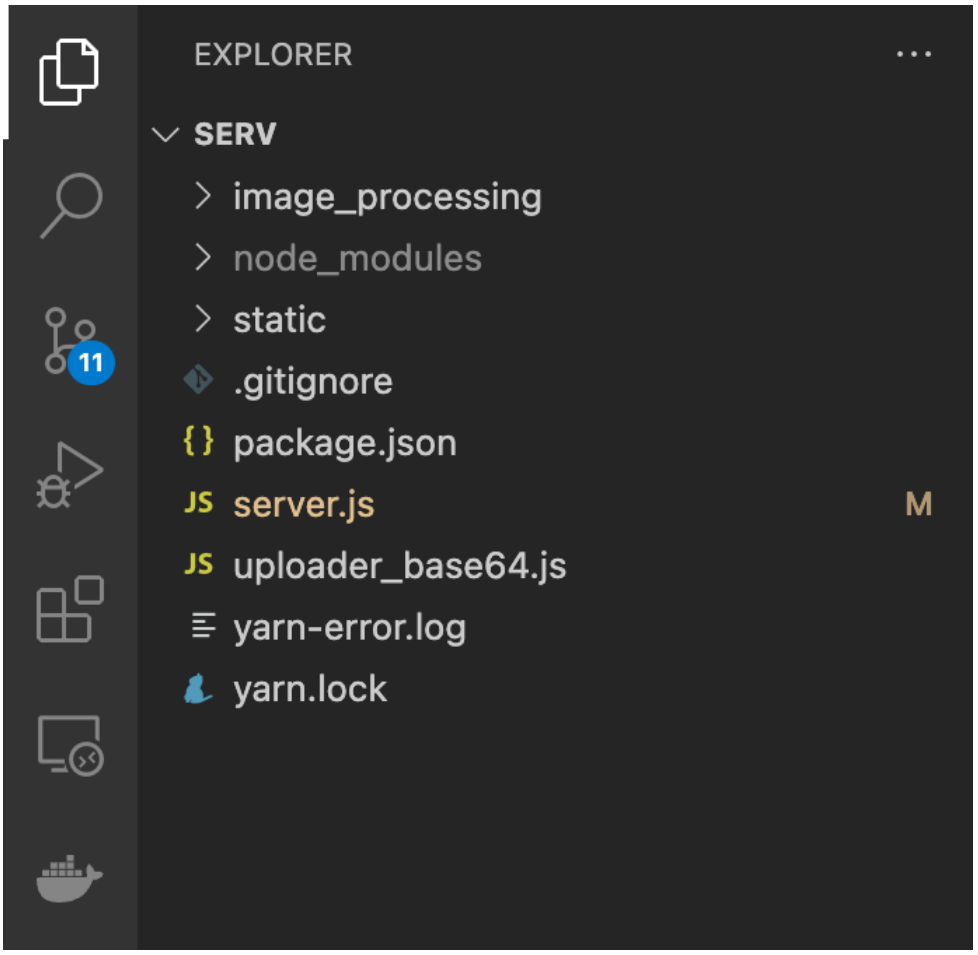

Рисунок 5 – Структура серверной части приложения

В node_modules располагаются пакеты, которые использовались в приложении и при разработке. Yarn-error и yarn.lock являются файлами пакетного менеджера yarn. То, от каких пакетов зависит работа программы описывается в файле package.json. В данном проекте используются:

1. Express – фреймворк для создания веб-приложений и API.
2. Body-parser – модуль Express, необходимый для извлечения тела из HTTP запроса.
3. Cors – модуль Express, обеспечивающий поддержку технологии CORS (Cross-origin resource sharing – совместное использование ресурсов между разными источниками).
4. Sharp – модуль Node.js, необходимый для обработки изображений

5. Uuid – модуль для создания строки по стандарту UUID (Universally unique identifier - универсальный уникальный идентификатор).
6. Is-base64 – пакет, содержащий функции верификации base64 строк.
7. Morgan – middleware, необходимый для логирования HTTP запросов.
8. Nodemon – утилита необходимая для автоматического перезапуска процесса.

Программа для обработки изображений располагается в директории image_processing. Сама программа состоит из двух файлов:

- image_functions – включает в себя функции, обозначенные на этапе определения требований к API. Они созданы при помощи пакета Sharp.
- image_processing_app – в нем представлен интерфейс для взаимодействия с image_functions, а также вспомогательные функции для работы с форматами ввода-вывода данных.

Функции редактирования изображений написаны в соответствии с use-case диаграммой. Как видно на рисунке 6, на строчках 32 и 33, функция изменения разрешения включает в себя вызов функции определения метаданных. Похожим образом устроена функция обрезки изображения.

```
27
28  async function resizeImage(buf, params) {
29      try {
30        return await sharp(img_buf)
31        .resize({
32          width: Math.round((await get_meta(img_buf)).width * params[i]),
33          height: Math.round((await get_meta(img_buf)).height * params[i])
34        })
35        // .toFormat("jpeg", { mozjpeg: true })
36        .toBuffer()
37        .then(data => data)
38      } catch (error) {
39        console.log(error);
40      }
41  }
42
```

Рисунок 6 – Функция изменения разрешения изображения

Стоит отметить, что функции для обработки изображений являются асинхронными и возвращают объект Promise. В связи с этим некоторые функции интерфейса взаимодействия с изображениями и обработчики запросов являются асинхронными. Для работы с Promise был использован современный специальный синтаксис языка JavaScript async/await[12].

Интерфейс взаимодействия является важной частью программы, так как sharp не позволяет работать с изображениями в формате base64. Функции интерфейса ответственны за перевод изображений из base64 в доступный для обработки тип данных языка JavaScript – Buffer. image_processing_app также помогает подготовить обработанные изображения к отправке.

В файле server Находится обработчик HTTP запросов, разработанный при помощи Express. В нем указаны конфигурационные данные сервера: информация о порте, на котором он работает, вызов пакетов cors и morgan. Обработка запросов к API осуществляется отдельно и расположена в файле uploader_base64. В соответствии с установленными ранее требованиями, на

рисунке 7, на 18 строке, API вынесен на отдельный, единый для всех его запросов, URL.

```
8   // App initiation
9   const app = express()
10  const port = 3030;
11
12  // Import form file uploader route
13  app.use(morgan('dev'));
14  app.use(cors({
15      origin: '*',
16      methods: ['GET','POST','DELETE','UPDATE','PUT','PATCH']
17  }));
18  app.use('/api', uploaded_base);
```

Рисунок 7 – Подключение вспомогательных пакетов и определение URL для API

В uploader_base64 описаны URL, по которым клиент может обращаться к API:

1. /api/ – получение метаданных
2. /api/rotate/ – поворот изображения
3. /api/resize – изменение расширения
4. /api/flip – отражение по вертикальной оси
5. /api/flop – отражение по горизонтальной оси
6. /api/crop – изменение размера
7. /api/gblur – изменение параметра размытия
8. /api/filter/ – применения фильтра, например /api/filter/greyscale применяет черно-белый фильтр

Ответ сервера может отличаться в зависимости от HTTP метода. При обращении по URL методом GET, клиенту отсылается JSON с отредактированным изображением. При отправке POST запроса на тот же URL, в соответствии с принципом кэширования REST [6], отредактированное

изображение сохраняется на сервере и в ответе отправляется JSON содержащий, помимо самого изображения, путь до него на сервере. Также, при обращении к API по /api/ методом DELETE, изображение с сервера удаляется.

Пример обработчика GET запроса представлен на рисунке 8.

```
65  router.get('/rotate', async (req, res) => {
66      if (!req.body.path) {
67          try {
68              imageProcessor.verifyBuffer(req.body.base64image)
69          }
70          catch(er) {
71              res.status(406).send(er)
72          }
73          const img = imageProcessor.verifyBuffer(req.body.base64image)
74          const newImg = await imageProcessor.changeImage(img, 'rotateImage', [Number(req.body.angle)])
75          const responseResult = await imageProcessor.getImgWithMeta(newImg, req.body.image_name)
76          res.status(200).json(responseResult)
77      }
78      else {
79          try {
80              imageProcessor.bufFromFile(req.body.path)
81          }
82          catch(er) {
83              res.status(400).send(er)
84          }
85          const img = imageProcessor.bufFromFile(req.body.path)
86          const newImg = await imageProcessor.changeImage(img, 'rotateImage', [Number(req.body.angle)])
87          const responseResult = imageProcessor.SaveImage(newImg, path.join(__dirname, './static/uploads'))
88          res.status(200).json(responseResult)
89      }
90
91  });
```

Рисунок 8 – обработчик GET запроса

Первоначально, при помощи пакета body-parser программа пытается извлечь путь к изображению из тела запроса. Если в запросе его не было, извлекается base64 строка. Она передается функции интерфейса программы обработки изображения для верификации. Затем, через тот же интерфейс строка переводится в тип Buffer, происходит редактирование изображения и формирование объекта, передаваемого в ответе. Если все прошло успешно, он форматируется в JSON и отправляется клиенту. В случае, если картинка уже была загружена ранее, обработчик выполняет все тот же набор действий, однако Buffer формируется уже на основе изображения, сохраненного на сервере. На строчках 71 и 83 также видно, что в случае какой-то ошибки, клиенту отправляется код состояния HTTP и дополнительная информация. Похожим

образом устроены обработчики всех остальных GET запросов. При POST запросе изображение сохраняется в директории /static/uploads.

Для тестирования API использовалось приложение Postman. Postman предоставляет возможности для:

- составления и отправки HTTP-запросов к API;
- создания набора последовательных запросов;
- структурированного хранения запросов;
- изменения заголовков и тела запросов;
- создания представления запроса в различных языках программирования и сторонних программах [8].

Исключительной особенностью Postman является интуитивно понятный интерфейс. Пример HTTP запроса в Postman представлен на рисунке 9

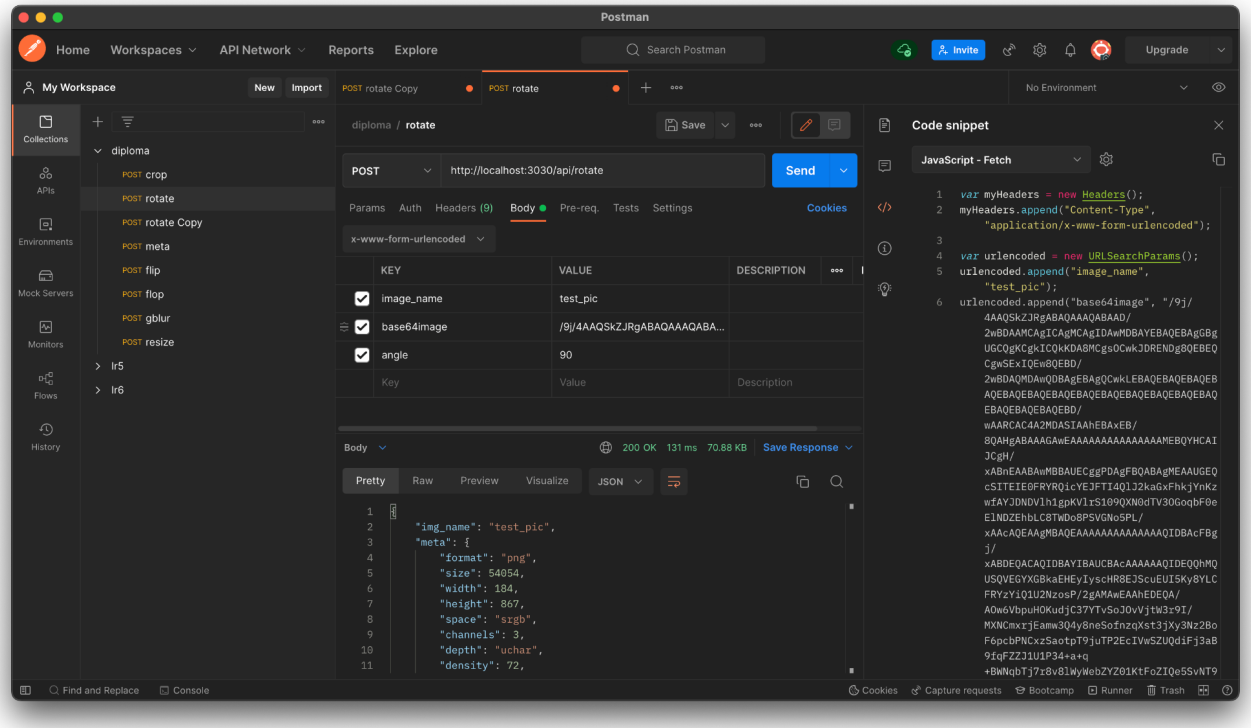

Рисунок 9 Пример HTTP запроса и ответа в Postman

В Postman были подготовлены и протестированы все возможные запросы к API. Тестирование также проводилось с клиентской части приложения.

2.4 Интеграция API

API был размещен средствами платформы Яндекс Облако. Для создания VDS, была приобретена виртуальная машина на ОС Ubuntu. Управление виртуальными машинами происходило в консоли Яндекс Облака. Через нее была составлена конфигурация машины. На Рисунке 10 представлен интерфейс изменения конфигурации виртуальной машины, на которой расположена серверная часть онлайн-редактора [13].

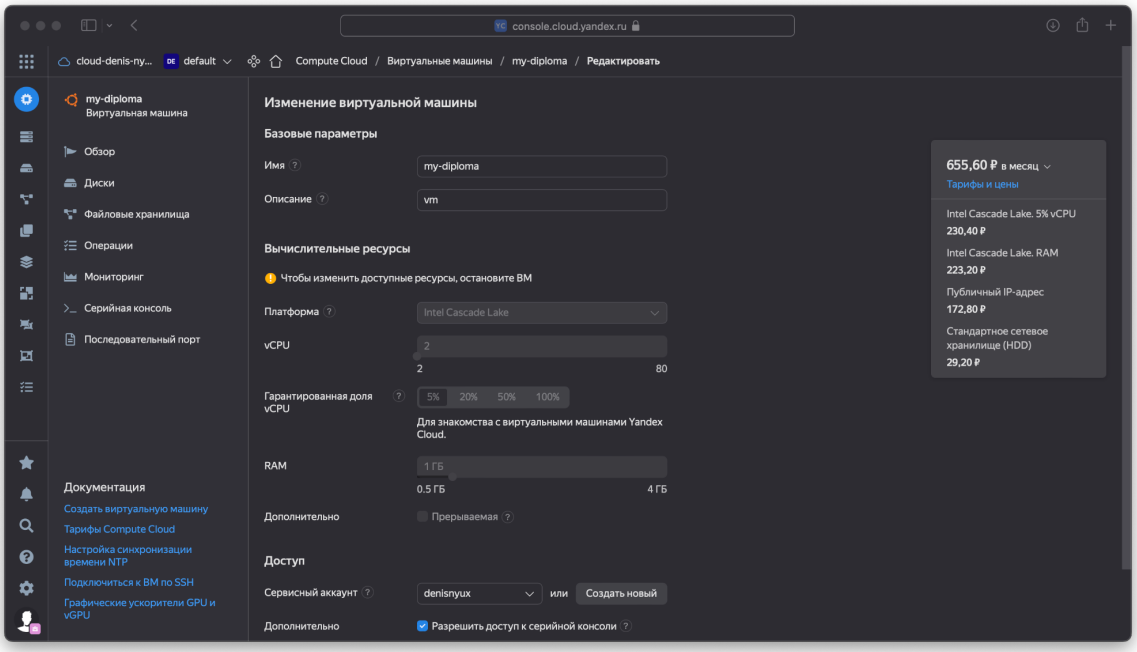

Рисунок 10 – Консоль Яндекс Облака

Для приложения, разрабатываемого в рамках данной дипломной работы, на VDS были выбраны следующие аппаратные характеристики:

- Процессор: Intel Cascade Lake
- Гарантированная доля vCPU: 5%
- Количество vCPU: 2

- RAM: 1 ГБ
- Объем дискового пространства: 10 ГБ

Для данной виртуальной машины был создан публичный IP адрес, при помощи которого удалось подключиться к ней по протоколу SSH (Secure Shell – безопасная оболочка)[17]. Была установлена платформа Node.js и при помощи утилиты SCP были перенесены исходные файлы приложения. Проводилась установка пакетов, необходимых для работы серверной части приложения, после чего оно было запущено. На рисунке 11 представлен интерфейс командной строки VDS, на котором было запущено приложение.

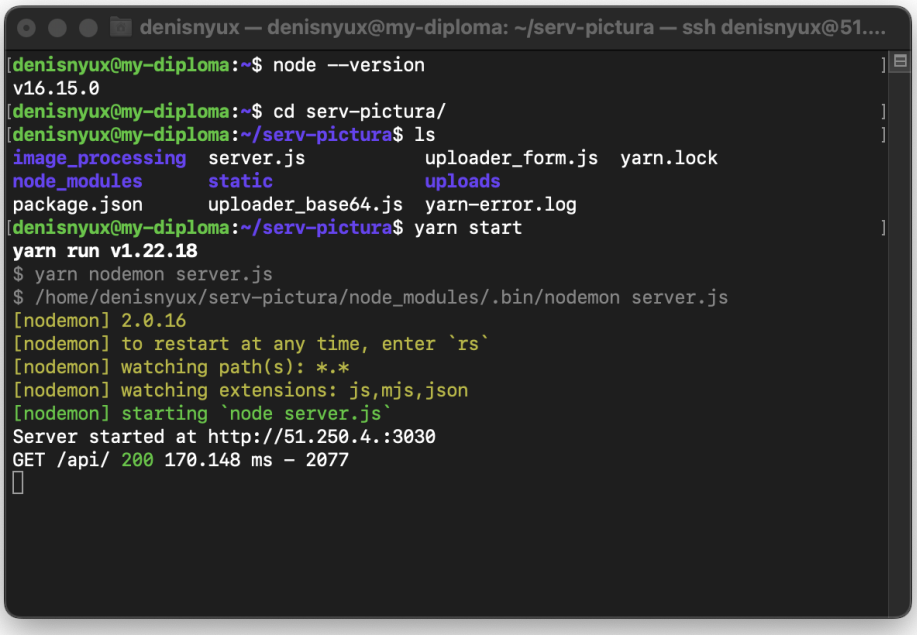

Рисунок 11 – Командная строка VDS

После запуска, оно было протестировано на работоспособность при помощи Postman.

Для дальнейшего размещения приложения в интернете, у доменного регистратора REG.RU было приобретено доменное имя palitra-redactor.ru.

Для настройки доменного имени VDS был использован сервис платформы Яндекс Облако – Cloud DNS. Этот сервис также доступен в консоли. В нем можно создать доменную зону и привязать ее к IP адресу выбранной виртуальной машины. Конфигурация доменной зоны, использованной при размещении приложения представлен на рисунке 12.

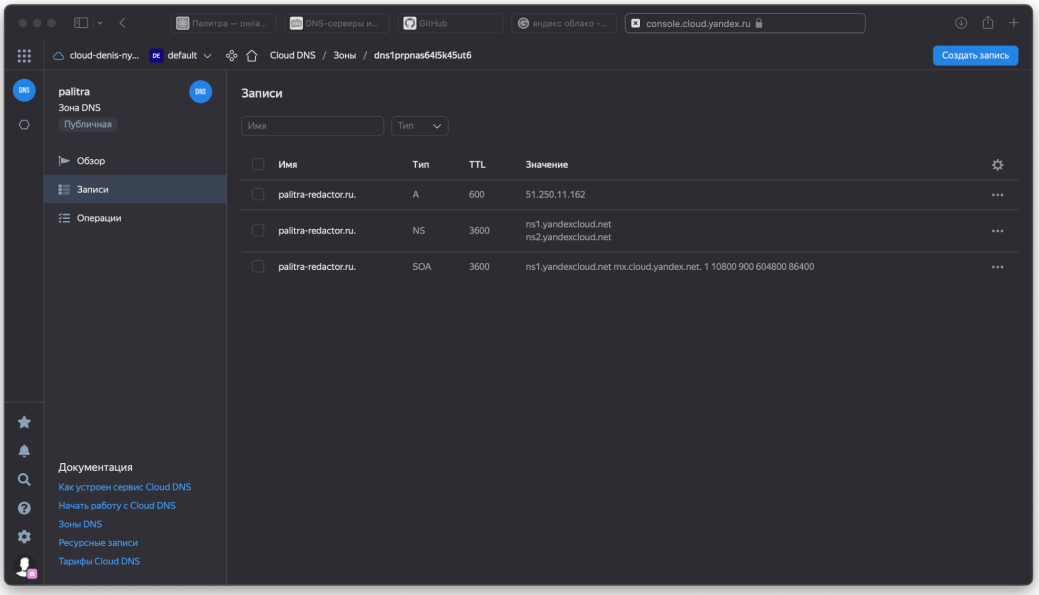

Рисунок 12 – Конфигурация доменной зоны

Через консоль управления доменными именами был настроен список DNS-серверов, обслуживающих этот домен. В консоли Яндекс Облака была создана DNS зона, в которую добавили ресурсные записи в зоне DNS.

Далее была проведена интеграция с клиентской частью приложения. Были предоставлены URL для обращения к API и проведены тесты совместной работы частей приложения. На рисунке 13 представлен пример реализации функций поворота и увеличения разрешения изображения, выполняющихся на серверной части онлайн-редактора.

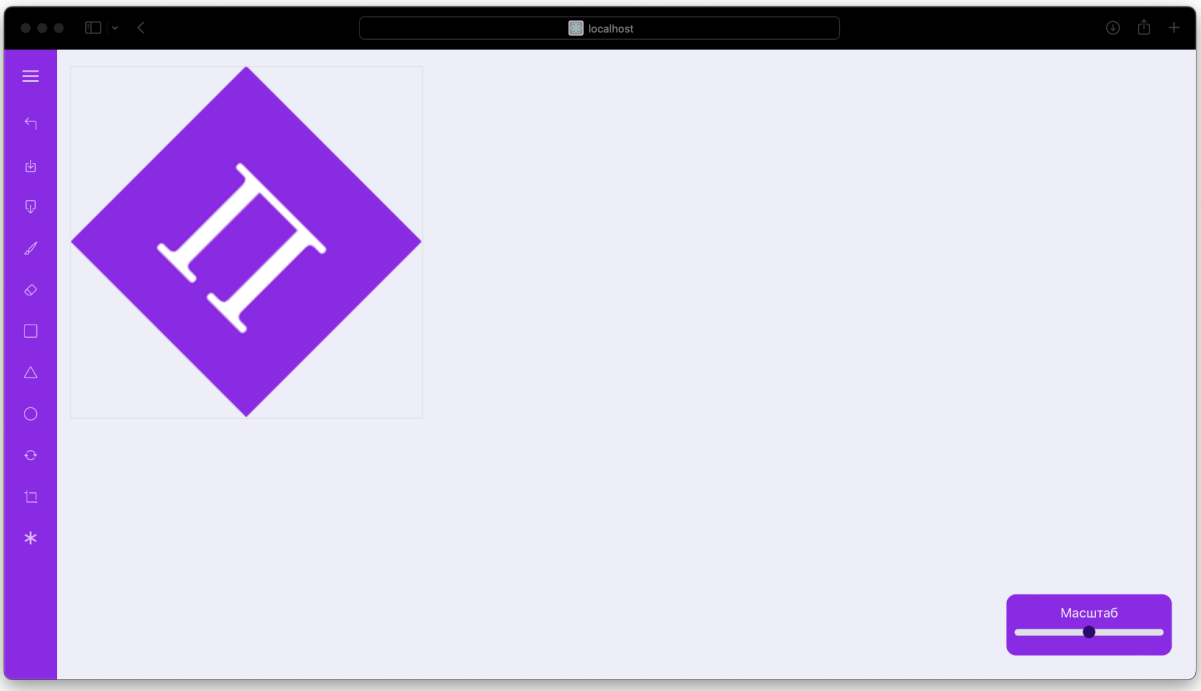

Рисунок 13 – Результат работы серверной части приложения со стороны клиентской

Для их реализации клиент отправлял следующую последовательность запросов:

1. POST запрос к URL pictura-redactor/api/rotate
2. GET запрос к URL pictura-redactor/api/rotate
3. POST запрос к URL pictura-redactor/api/resize
4. GET запрос к URL pictura-redactor/api/resize

Выводы по главе 2:

1. На основе выводов из главы 1 и технического задания, составленного в ходе преддипломной практики, были определены требования к разрабатываемому продукту.
2. В ходе проектирования API были разработаны use-case диаграмма и блок-схема приложения.
3. В соответствии с поставленной задачей и требованиями была разработана серверная часть приложения и API по редактированию изображений;

4. В соответствии с поставленной задачей на виртуальном выделенном сервере был размещен код серверной части приложения.

ЗАКЛЮЧЕНИЕ

Цель разработать API серверной части онлайн-микросервиса по редактированию изображений на основе импортозамещенных веб-технологий, которая была поставлена в данной выпускной квалификационной работе, была выполнена.

В процессе выполнения данной работы были решены следующие задачи:

1. Осуществлен анализ современных подходов к написанию онлайн сервисов и API для обработки изображений.
2. Был проведен сравнительный анализ отечественных сервисов, предоставляющих услуги аренды виртуальных выделенных серверов;
3. Был определен набор функций, которые выполняются на серверной части сервиса обработки изображений;
4. Была разработана серверная часть приложения и API, которые реализуют функции по редактированию изображений;
5. Код серверной части приложения был размещен на виртуальном выделенном сервере.

СПИСОК ЛИТЕРАТУРЫ

1. Интернет-справочник MDN Web Docs: [сайт]. URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview (дата обращения: 20.05.2022) – Режим доступа: свободный.
2. Интернет-справочник W3Docs: [сайт]. URL: https://ru.w3docs.com/uchebnik-html/metody-http-zaprosa.html (дата обращения: 20.05.2022) – Режим доступа: свободный.
3. Ресурс World Wide Web Consortium W3: [сайт]. URL: https://www.w3.org/TR/soap/ (дата обращения: 20.05.2022) – Режим доступа: свободный.
4. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. 2-е издание. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бест- селлеры O’Reilly»). ISBN 978-5-4461-0590-8
5. Amundsen, Mike., Ruby, Sam., Richardson, Leonard. RESTful Web APIs: Services for a Changing World. Соединенные Штаты Америки: O'Reilly Media, 2013.v
6. Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.
7. ГОСТ Р 51904-2002 Программное обеспечение встроенных систем. Общие требования к разработке и документированию – Официальное издание. М.: Стандартинформ – 2005 год.
8. Publishing.Westerveld, Dave. API Testing and Development with Postman: A Practical Guide to Creating, Testing, and Managing APIs for Automated Software Testing. Соединенные Штаты Америки: Packt Publishing, 2021.
9. API обработки изображений Pixo: [сайт]. URL: https://pixoeditor.com/documentation/editing-api/ (дата обращения: 20.05.2022) – Режим доступа: свободный.

10. API обработки изображений Pixelixe: [сайт]. URL: https://pixelixe.com/docs/v2/image-processing.html#image-processing-api (дата обращения: 20.05.2022) – Режим доступа: свободный.
11. Документация Node.js модуля Sharp: [сайт]. URL: https://sharp.pixelplumbing.com/performance (дата обращения: 20.05.2022) – Режим доступа: свободный.
12. Статья Modern Asynchronous JavaScript with Async and AwaitNodejs: [сайт]. URL: https://nodejs.dev/learn/modern-asynchronous-javascript-with-async-and-await (дата обращения: 20.05.2022) – Режим доступа: свободный.
13. Документация Yandex Compute Cloud: [сайт]. URL:https://cloud.yandex.ru/docs/compute/
14. Чепегин Илья Дмитриевич Серверный JavaScript - преимущества и недостатки node. JS // Вестник науки и образования. 2020. №12-1 (90). URL: https://cyberleninka.ru/article/n/servernyy-javascript-preimuschestva-i-nedostatki-node-js (дата обращения: 23.05.2022).
15. RFC 4648. The Base16, Base32, and Base64 Data Encodings URL: https://datatracker.ietf.org/doc/html/rfc4648 (дата обращения: 20.05.2022) – Режим доступа: свободный.
16. Bassett, Lindsay. Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. Тайвань: O'Reilly Media, 2015.
17. Ракчеев Алексей Юрьевич Работа протокола ssh на практике // Научный журнал. 2020. №3 (48). URL: https://cyberleninka.ru/article/n/rabota-protokola-ssh-na-praktike (дата обращения: 23.05.2022).
18. Уткина Любовь Ивановна Возможности виртуального выделенного сервера в поддержке сайта компании // Огарёв-Online. 2017. №2 (91). URL:

https://cyberleninka.ru/article/n/vozmozhnosti-virtualnogo-vydelennogo-servera-v-podderzhke-sayta-kompanii (дата обращения: 23.05.2022).

19. Дроздов Сергей Анатольевич, Луканина Василиса Евгеньевна Особенности проектирования серверного и клиентского программного обеспечения web-сайта с использованием rest-архитектуры // Вестник МГУП. 2016. №2. URL: https://cyberleninka.ru/article/n/osobennosti-proektirovaniya-servernogo-i-klientskogo-programmnogo-obespecheniya-web-sayta-s-ispolzovaniem-rest-arhitektury (дата обращения: 23.05.2022).
20. Internet Media Type registration, consistency of use. W3C. URL: https://www.w3.org/2001/tag/2002/0129-mime (дата обращения: 09.05.2022) – Режим доступа: свободный.

ПРИЛОЖЕНИЕ А

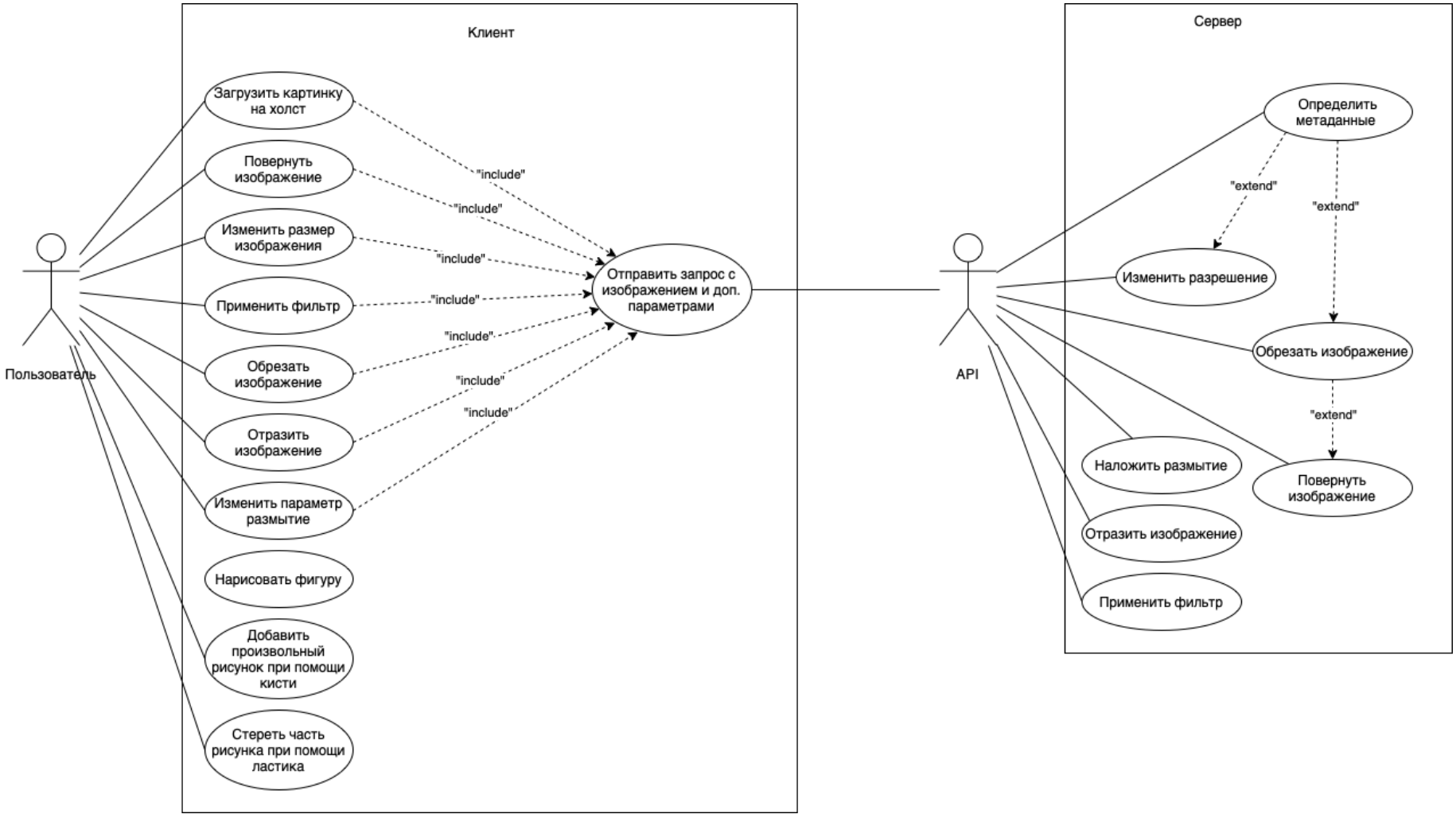

Рисунок 1 Use-case диаграмма приложения