**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

РОССИЙСКОЙ ФЕДЕРАЦИИФедеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**ОТЧЕТ о выполнении**

**ЛАБОРАТОРНОЙ РАБОТЫ №1**

**“Распознавание текста на пользовательском рисунке с использованием облачного сервиса Yandex Cloud”**

**по дисциплине “Основы облачных технологий”**

Работу выполнил студент 4 курса 4ИТ гр. \_\_\_\_\_\_\_\_\_\_\_ Меликов Р.М.

Работу проверил канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ПолупановА

Краснодар 2023

Тема: Опознование текста на рисунках с использованием облачных технологий Yandex Cloud

Цель: Изучение теоретических основ облачных вычислений, внутренней структуры и практической реализации, и прикладных примеров использования облачных вычислений и веб-сервисов.

Задание: Разработка облачного веб-приложения для распознавания текста на рисунке

Постановка задачи: разработать веб-приложение, которое принимает рисунок пользователя на холсте и распознает текст на нем с использованием возможностей Yandex Cloud

# Подготовка интерфейса приложения

В разрабатываемом веб-приложении предполагается использование формы, на которой пользователь через веб-браузер рисует надпись и выбирает язык распознавания – русский или английский. В файле index.html находится простая верстка формы, а в файле style.css хранятся стили, а в main.js – основные скрипты для взаимодействия с облачными функциями и интерфейсом. Для написания кода будем работать в редакторе Visual Studio Code.

В head звене файла html указывается ссылка на ссылаемые объекты, о которых было указано выше, указывается название страницы как «Распознавание» и определяется распознавание границ окон в приложении.

В body хранится заголовочный текст, выводящий на окно в браузере «Распознавание текста» большими жирными буквами, и блочный элемент div, в котором и хранится сама форма непосредственно. В форме содержатся объект canvas - холст определенного размера, на котором пользователь может рисовать надписи на распознавание, прямо под холстом находится кнопка «Отчистить рисунок», которая позволяет полностью отчистить холст при нажатии на нее. Дальше в форме находится выпадающий список select, в котором можно выбрать язык, на котором нарисован текст, что необходимо для распознавания, а также специальный конейтнер под *Yandex SmartCaptcha*, про который будет рассказано позже, и кнопка отправки, которая отключена по умолчанию, и включается лишь при успешном прохождении капчи.

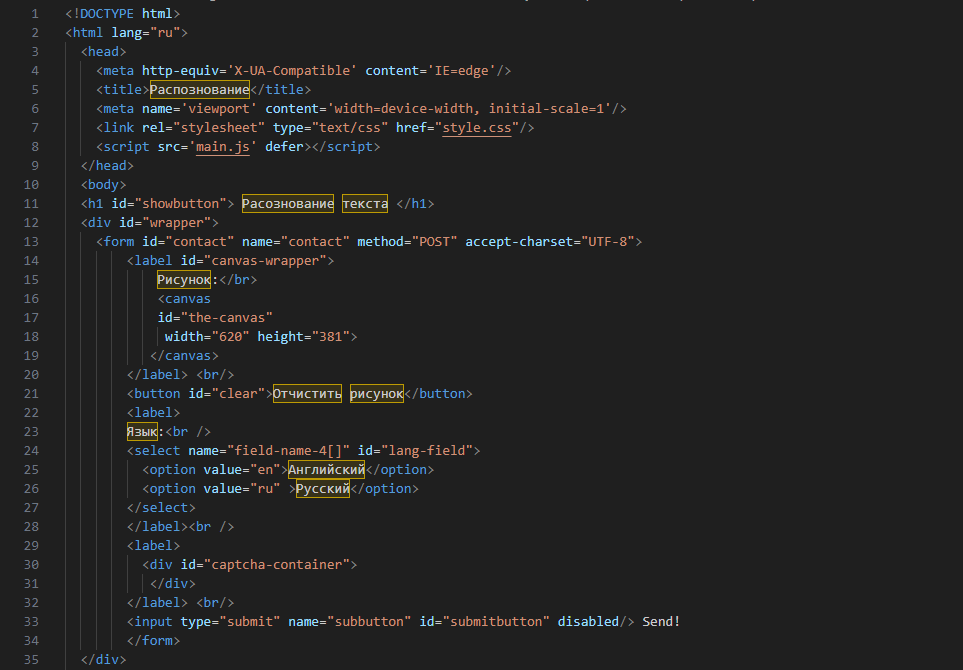
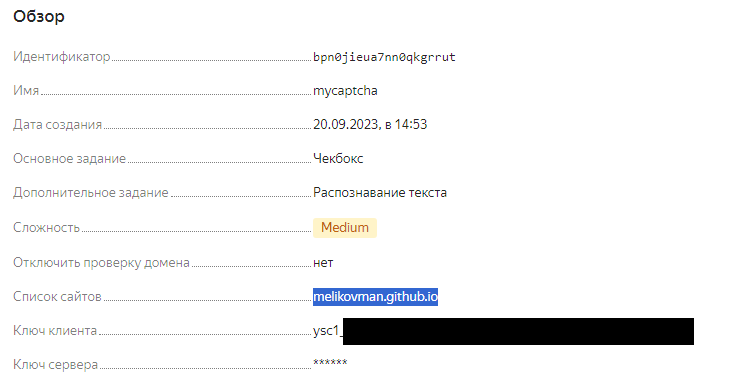
Сайт разворачивается на домене <https://MelikovMan.github.io/cloud> - с использованием сервиса хостинга статических сайтов *Github Pages*

Рисунок 1 – форма HTML

# Подключение Yandex SmartCaptcha к форме отправки

После создания рабочего платежного аккаунта и запуска тестового периода, для создания Captcha для подключения необходимо выполнить следующие действия в консоли управления YandexCloud:

1. заходим в каталог default и в списке всех услуг выбираем Yandex SmartCaptcha,
2. в появившимся окне необходимо нажать на кнопку *«Создать капчу»,*
3. в появившемся окне выбрать имя (в данном случае выбрано mycaptcha), список разрешенных доменов (melikovman.github.io), дополнительно указать внешний вид и сложность и нажать на кнопку *«Создать».*

Рисунок 2 – окно Yandex SmartCaptcha

В следующем окне необходимо запомнить поле ключа клиента. После этого необходимо использовать специальный скрипт для подгрузки функционирования капчи.

В первую очередь используется элемент script, подгружающий капчу с серверов Яндекса. Параметр render указывает, что элемент загружается лишь после завершения загрузки страницы, а в onload – функция JavaScript, которая вызывается после завершения рендера SmartCaptcha.

В дальнейшем скрипте определяется функция callback, которая включает кнопку отправку формы только при успешном прохождении капчи, и отключает при всех остальных случаях. SmartCaptchaInit() – функция, которая вызывается после завершения загрузки страницы, которая рендерит в элемент DOM с id captcha-container капчу, с указыванием функции обратного вызова при обработке, а также ключа клиента, полученный ранее, а также функции сброса капчи и получения ответа.

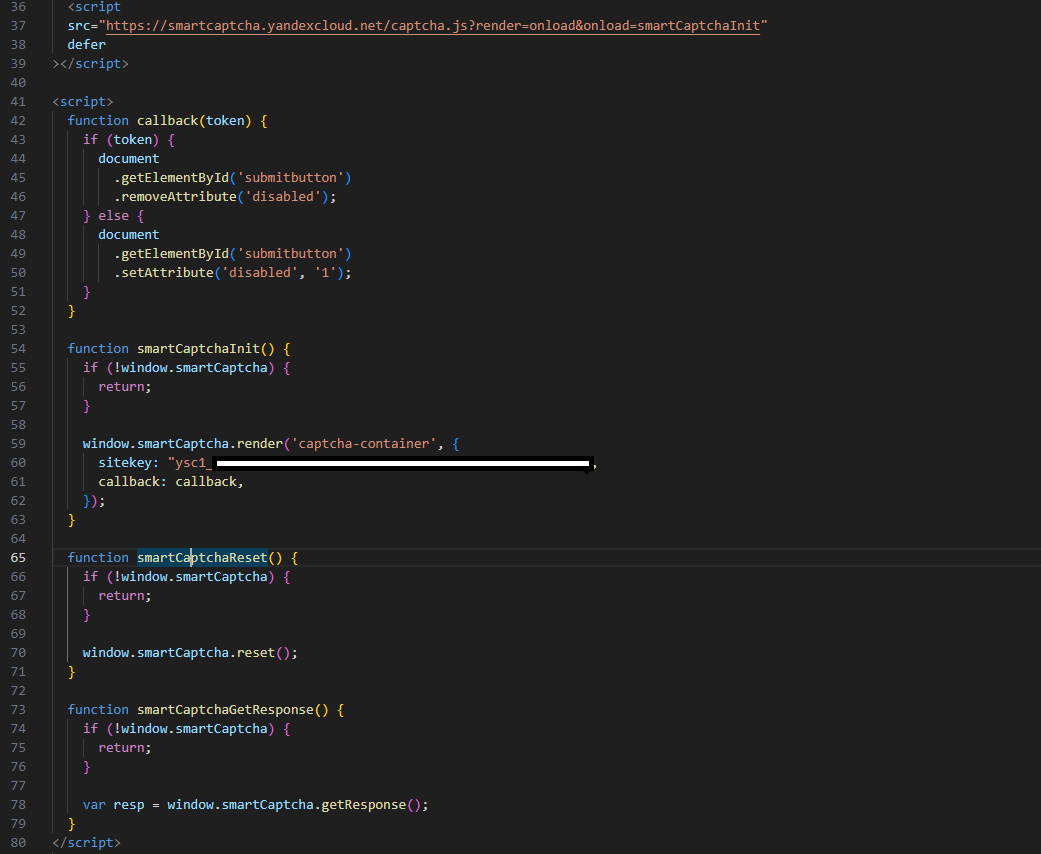


Рисунок 3 – скрипт для подгрузки капчи в элемент

В файле style.css указвает стили, осуществляющее выравнивание элементов по центру с использованием flexbox, а в main.js указывается обработчки события касания и мыши для рисования, отправки запроса в облачную функцию и обработки результатов. В теле запроса HTTP методом POST содержится JSON-файл, содержащий свойства language – язык, который распознается, и base64 – рисунок пользователя, закодированный в base64. Результат успешной обработки выводится ниже формы, если обработка завершена, иначе выводится сообщение о том, что сообщение не распознано.

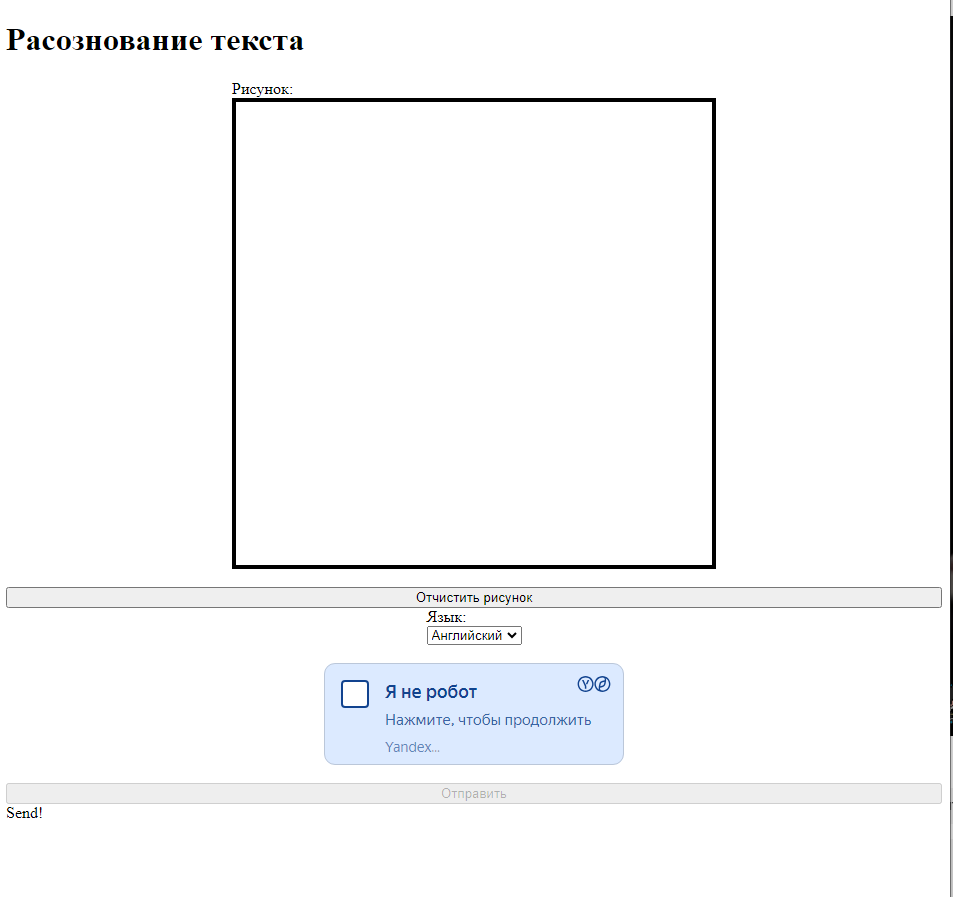


Рисунок 4 – готовый интерфейс

# Создание сервисного аккаунта и ключа API для работы с Vision API

Для работы с Vision API – нейронной сетью для распознования текста необходимо создать сервисный аккаунт на платформе Yandex Cloud с нужной ролью и сформировать для него ключ API. Необходимо выполнить следующие действия в консоли управления:

1. вверху выбрать вкладку *«Сервисные аккаунты»*,
2. нажать на кнопку в правом верхнем углу *«Создать сервисный аккаунт»*,
3. во всплывшем модальном окне указать имя сервисного аккаунта, его описание и присвоить сервисному аккаунту роли, обязательно для него указать роль *ai.vision.user,* после чего нажать на кнопку «*Создать*»;
4. после создания аккаунта он должен появится в списке сервисных аккаунтов, требуется выбрать этот аккаунт по имени из списка,
5. в полученном окне необходимо нажать на кнопку *«Создать новый ключ»,* которая находится в верхнем левом углу,после чего во сплывающем окне появится идентификатор ключа и сам скрытый ключ API,
6. сохраняем ключ API для дальнейшего использования и закрываем окно, обратите внимание, что ключ после закрытия окна становится недоступным.

# Облачная функция.

Для создания облачной функции обработки запроса, поступившего от веб-интерфейса, создается облачная функция

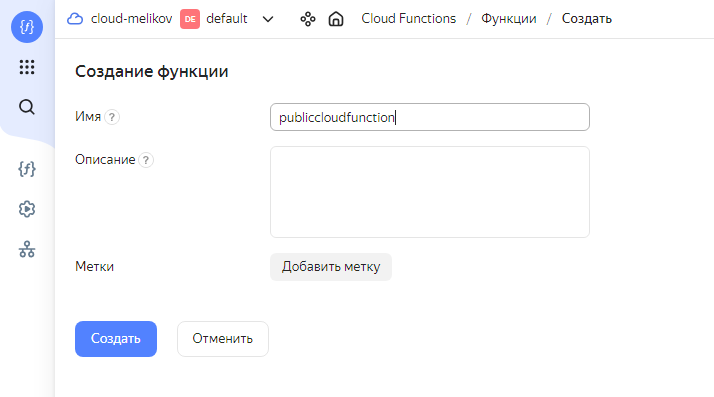
1. в консоли управления во вкладке директории среди всех функций выбирается раздел Cloud Functions,
2. в появившемся окне необходимо нажать на кнопку *«Создать функцию»,*
3. указать имя функции и нажать на кнопку *«Создать»*

Рисунок 5 – окно создания облачной функции

После создания облачной функции появляетсе меню функции. Для того, чтобы к функции можно было обратиться без аутентификации пользователя ее необходимо сделать публичной. Для этого необходимо нажать на рубильник «Публичная функция»

# Работа с редактором кода

В главной меню облачной функции необходимо перейти во вкладку «Редактор», которая находится справа. В открывшемся окне выбираем среду выполнения Node.js/16 и способ «Редактор кода». Данная опция автоматически создает обработчик событий с Node.js. В опциях снизу можно определить память, выделяемую для обработки функции, максимальное время ответа, а также переменные окружения. Для данной функции время ожидания в 10 секунд и максимальную память в 256 Мб.

В данной функции будут использоваться внешняя зависимость node-fetch – реализация асинхронных запросов HTTP. Для разрешения этих зависимостей в Node.js используется менеджер пакетов npm. Облачные функции автоматически подгружают зависимости из файла package.json и package-lock.json

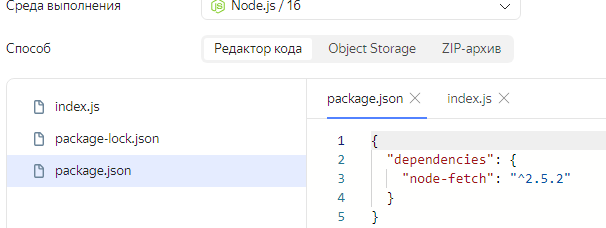


Рисунок 6 – зависимости в package.json (package-lock.json аналогичен)

В файле index.js реализуется сама функция обработки запроса, который будет поступать от веб-интерфеса. Вначале подгружается зависимость node-fetch, а в самой функции обработке handler, которая является асинхронной, в переменной event хранится сам запрос, а в context – метаданные функции и обращения к ним. Помимо этого, для работы с Vision API, необходимо знать идентификатор каталога, который можно выяснить на главной странице каталога, он находится вверху рядом с названием, а также ключ API пользователя, полученный в пункте 2.

Поскольку тело запроса содержит JSON, закодированный в b64, он преобразуется в буфер битов, который преобразуется в строку, а затем в объект JavaScript. Используя данные, полученные из запроса и преобразованные в объект, формируется объект bodyImage, который является телом запроса к REST API Vision. В bodyImage указывается идентификатор каталога, а также специфика анализа – изображение в формате b64, выполняемая функция (TEXT\_DETECTION – распознавание текста), а также используемый язык. После формирования объекта, асинхронно выполняется запрос на REST API Vision, используя функцию batchAnalyze, метод POST, а в заголовках указывается, что мы отправляем объект типа JSON, принимаем JSON, а также API ключ для аутентификации в сервисе. После этого, результат преобразовывается в объект JavaScript и определяется как тело запроса, а статус запроса – 200.

Веб интерфейс выделяет из полученного ответа тело, если не было ошибок и код запроса равен 2xx, то выделяет из тела ответа распознанный текст и создает DOM элемент с распознанным текстом.



Рисунок 7 – результат распознавания нарисованного текста

# Приложение 1

Файл style.css

#the-canvas{

display:block;

border: 4px solid black;

}

#contact{

display: flex;

flex-direction: column;

}

#contact label{

display: block;

margin: auto;

}

# Приложение 2

Файл main.js

window.addEventListener('DOMContentLoaded',(e)=>{

let centerCanvasPosition = function(){

canvas.width = window.innerWidth / 2;

canvas.height =window.innerHeight / 2;

};

let clearCanvas = function(e) {

e.preventDefault();

ctx.fillStyle='white';

ctx.fillRect(0,0,canvas.width,canvas.height);

}

let pos = {x:0,y:0};

let getCanvasRelative = function (e) {

let canvas = e.target,

bx = canvas.getBoundingClientRect();

let x = (e.changedTouches ? e.changedTouches[0].clientX : e.clientX) - bx.left,

y = (e.changedTouches ? e.changedTouches[0].clientY : e.clientY) - bx.top;

pos.x=x;

pos.y=y;

};

let drawMouse = function(e) {

if (e.buttons !== 1) return;

ctx.beginPath(); // begin

ctx.lineWidth = 5;

ctx.lineCap = 'round';

ctx.strokeStyle = '#c0392b';

ctx.moveTo(pos.x, pos.y); // from

getCanvasRelative(e);

ctx.lineTo(pos.x, pos.y); // to

ctx.stroke();

};

let canvas = document.getElementById('the-canvas'),

ctx = canvas.getContext('2d');

canvas.addEventListener('mouseup',getCanvasRelative);

canvas.addEventListener('mousedown',getCanvasRelative);

canvas.addEventListener("touchstart", getCanvasRelative);

canvas.addEventListener("touchend", getCanvasRelative);

canvas.addEventListener("mousemove",drawMouse);

canvas.addEventListener("touchmove",drawMouse);

let clear = document.getElementById('clear');

clear.addEventListener("click",clearCanvas);

window.addEventListener('resize',centerCanvasPosition);

const form = document.getElementById("contact")

form.addEventListener("submit",(e)=>{

e.preventDefault();

let select = document.getElementById("lang-field");

let obj = {

base64: canvas.toDataURL().replace('data:', '').replace(/^.+,/, ''),

language: select.options[select.selectedIndex].value

}

document

.getElementById('submitbutton')

.setAttribute('disabled', '1');

fetch("https://functions.yandexcloud.net/d4et655ongl4sskem62s",

{

method: "POST",

body: JSON.stringify(obj)

})

.then(function (response) {

if (!response.ok) {

throw new Error(response);

}

return response;

})

.then((response) => {

alert("Success");

return response.json();

}).then((results) => {

//console.log(results);

//console.log(results.results[0].results[0].textDetection);

const textarr = new Array();

const textResults = results.results[0].results[0].textDetection;

textResults.pages?.forEach((page) =>

page.blocks?.forEach((block) =>

block.lines?.forEach((line) =>

line.words?.forEach((word) =>

textarr.push(word.text))

)

)

);

let textNodeQuery = document.getElementById("recognized-text");

if (!textNodeQuery) {

let link = document.createElement("a");

let wrap = document.getElementById("wrapper");

let textNode = document.createElement("div");

textNode.id = "recognized-text";

textNode.innerHTML = (textarr.length === 0) ? "Not recognized" : `Recognized: ${textarr.join(" ")}`;

link.id = "bucket-link";

link.innerHTML = "Download JSON";

link.href = `https://testbucker.hb.ru-msk.vkcs.cloud/${results.link}.json`;

textNode.appendChild(document.createElement("br"));

textNode.appendChild(link)

wrap.appendChild(textNode)

}

else {

textNodeQuery.innerHTML = (textarr.length === 0) ? "Not recognized!" : `Recognized: ${textarr.join(" ")}`;

}

})

.catch((error) => {

alert("Failure");

console.log(error);

}).finally(() => document

.getElementById('submitbutton')

.removeAttribute('disabled'));

})

})

# Приложение 3

Файл index.js

const fetch = require("node-fetch");

module.exports.handler = async function (event, context) {

const folderIdi = "b1g9s3shh…"(hidden)

const apikey = "AQVN173…"(hidden)

const formData = JSON.parse(Buffer.from(event.body, 'base64').toString())

const bodyImage = {

folderId: folderIdi,

analyze\_specs: [{

content: formData.base64,

features: [{

type: "TEXT\_DETECTION",

text\_detection\_config: {

language\_codes: formData.language

}

}]

}]

}

let response = await fetch("https://vision.api.cloud.yandex.net/vision/v1/batchAnalyze",

{

method:"POST",

headers:

{

"Content-type": "application/json",

"Accept": "application/json",

"Authorization": `Api-Key ${apikey}`

},

body: JSON.stringify(bodyImage)

});

let bodyResponse = await response.json();

return {

statusCode: response.status,

body: bodyResponse,

};