Федеральное государственное бюджетное образовательное учреждение высшего образования “МИРЭА – Российский технологический университет”

Институт перспективных технологий и индустриального программирования

# Практическая работа на тему:

“Разработка web приложения для поиска и обработки данных о вакансиях с различных источников в открытом доступе”

|  |  |
| --- | --- |
| Выполнили:  Ворсин К.А.  Наротнев Г.В.  Викторов А.Д.  Гуськов Н.  Бурлак М.А. | Проверил:  Макиевский С. Е. |

Москва 2024 г.

# Оглавление

[Введение 3](#_TOC_250006)

[Глава 1. Теоретическая часть 4](#_TOC_250005)

* 1. Описание программируемой системы 4
  2. Методология разработки 4
  3. Обзор существующих решений 5
  4. Прототипирование и дизайн 5
  5. Требования к программируемой системе 7
  6. Метод и технология тестирования программируемой системы 7

[Глава 2. Технологическая часть 8](#_TOC_250004)

* 1. Диаграммы 8

[Глава 3. Практическая часть 11](#_TOC_250003)

* 1. Дизайн прототип 12
  2. Реализация требований к системе 12
  3. Функциональное тестрирование программного продукта 12
  4. Инструкция по эксплуатации 12

[Заключение 14](#_TOC_250002)

[Список используемых источников 15](#_TOC_250001)

[Приложения 15](#_TOC_250000)

Приложение А – apiOauth.js 15

Приложение B – requests.js 15

Приложение C – scheduler.js 15

Приложение D – server.js 15

# Введение

На протяжении всего срока разработки у членов нашей команды были следующие основные роли, распределенные по степени умений индивида в той или иной сфере:

Ворсин К.А – Тимлид

Наротнев Г.В. – Frontend разработчик Гуськов Н. – Backend разработчик Бурлак М.А. – Бизнес аналитик

Викторов А.Д. – Тестировщик

Наша команда поставила перед собой задачу разработки web-приложения, которое посредством интеграции с API самых популярных сайтов по поиску работы, будет составлять таблицу с подсчетом навыков и технологий, а также сортировать результаты по частоте упоминания потенциальным нанимателем. Целью практической работы является разработка такого web-приложения.

**Цель работы:** Разработка web приложения для поиска и обработки данных о вакансиях с различных источников в открытом доступе

## Задачи работы

* + 1. Описать программируемую систему.
    2. Рассмотреть существующие решения-аналоги по данной, либо смежной темам.
    3. Сформировать требования к программируемой системе.
    4. Спроектировать необходимые диаграммы.
    5. Реализовать программный продукт в соответствии с требованиями.
    6. Провести функциональное тестирование программного продукта.
    7. Составить инструкцию по использованию программного продукта.
    8. Составить отчет по работе.
    9. Сдать отчет и представить его к защите.

**В качестве основных методов исследования** применены анализ, синтез, сравнение и

моделирование. Практическая реализация поставленной задачи соответствует основным подходам к разработке программного обеспечения.

**Информационной базой** исследования являются открытые источники, в том числе доступные в сети Интернет, а также материалы курса «Технологии индустриального программирования», доступные через систему дистанционного обучения РТУ МИРЭА.

В данном отчете будет представлен процесс разработки программного продукта, в том числе теоретический обзор области и системы, технологическое проектирование и описание системы. а также непосредственно результаты разработки.

# Глава 1. Теоретическая часть

* 1. Описание программируемой системы

Web-приложение по поиску самых популярных навыков и технологий и их сортировки предназначено для начинающих специалистов, в особенности планирующих работать в сфере IT. Оно позволит на старте изучения ознакомиться с технологиями которые чаще всего используются в коммерческой разработке приложений, систем и других компонентов. Основные компоненты клиентской части системы:

* + - Диалоговое окно с кнопками и функциональным поиском.
    - Ведение заметок: создание структурированных текстовых заметок.
    - Управление файлами: загрузка, хранение и организация файлов в облачном пространстве.
    - Безопасность: авторизация и аутентификация с использованием JWT, защита данных пользователей.

Также организована имитация серверной части приложения, которая позволит далее расширять функционал и значительно сократит время взаимодействия модулей организующих обработку данных и вывод их на клиентскую часть сервиса. Среди ее компонентов следующие:

* + - server – модуль который организует запуск сервера, получает информацию о портах и запросах.
    - apiOauth – модуль аутентификации и авторизации всего приложения к сторонним API.
    - requests – расширяемый модуль управления запросами и обработки результатов для дальнейшего вывода на клиентской части.
    - sheduler – модуль генерации запланированных запросов работающий с большими данными и отвечающий за их загрузку в локальное хранилище.
    - buffer – директория для хранения результатов запросов обработанных сервером изолировано от клиентской части, представленных в упрощенном для вывода формате.
  1. Методология разработки

Методология Agile — это популярный подход, в котором основное внимание уделяется гибкости, сотрудничеству и оптимизации процессов для реализации качественного проекта. Это итеративный подход, и приоритет в нем отдается обратной связи от владельца продукта и адаптации к изменяющимся требованиям. Цикл разработки ПО по Agile-методологии можно разбить на шесть этапов: планирование, проектирование, разработка, тестирование, развертывание и обслуживание.

Основные принципы Agile:

1. Люди и взаимодействие важнее процессов и инструментов. приоритетом, что способствует улучшению коммуникации.
2. Работающий продукт важнее исчерпывающей документации — команды фокусируются на создании функционирующих частей продукта, которые можно тестировать и внедрять.
3. Сотрудничество с заказчиком важнее согласования условий контракта — постоянная обратная связь от заказчика помогает корректировать курс и удовлетворять текущие потребности.
4. Готовность к изменениям важнее следования первоначальному плану — Agile позволяет быстро менять приоритеты в зависимости от новых данных или требований.

Мы выбрали Agile, в частности подход Kanban, потому что все из нас начинающие программисты с различным набором профессиональных навыков и личных качеств. Также при разделении задач было важно чтобы каждый мог взять ответственность за другого, а своей частью кода не затронуть функционал другого. Постоянное введение команды в прогресс разработки разных уровней приложения формировало правильную общую картину о текущем состоянии разрабатываемой системы и подчеркивало важность каждого отдельного задания.

* 1. Обзор существующих решений

Анализ аналогичных решений показал, что на рынке существуют следующие продукты:

1. Hh Карьера — собственный разностороний сервис помощи в поиске работы, в том числе имеющий информацию о навыках, указывает их косвенное влияние на уровень зарплаты. Недостатки: конкретный раздел с навыками довольно трудно найти на сайте, сервис не предоставляет точного количества вакансий по указанным навыкам, информация о влиянии владения навыком на заработную плату имеет псевдодоказательную базу, в связи с тем что многие оставляют графу оклада пустой, указывают дополнительно другие навыки в требованиях.
2. easyoffer — сервис для подготовки к собеседованиям, предоставляющий услуги менторства, подсчитывающий упоминания навыков, по схожим с нашим функционалом алгоритмам. Недостатки: устаревший, неприятный глазу дизайн, явно ограниченный перечень IT вакансий, не дает возможности посредством поиска сразу найти нужную вакансию из перечня. ограничен в расширении имеющихся функций.

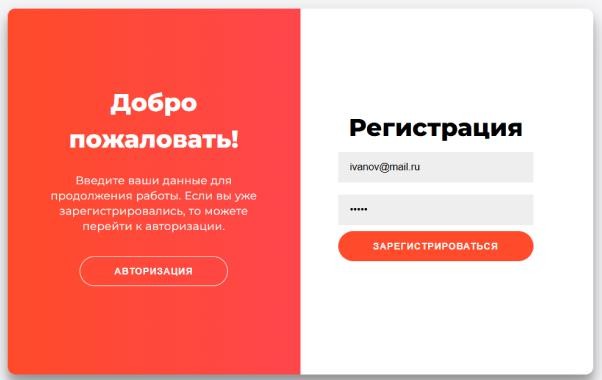
Наше приложение делает акцент на качественную информацию, исключает напрямую не касающийся основной цели функционал, удобно как уже определившемуся, так и неопределившемуся с будущей работой пользователю.

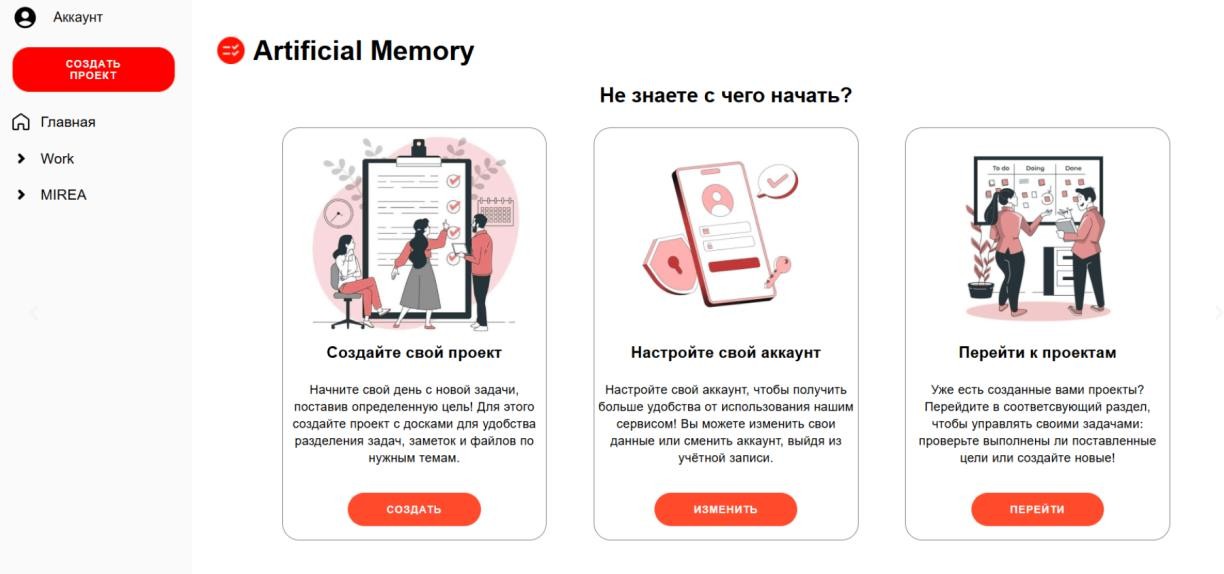
* 1. Прототипирование и дизайн

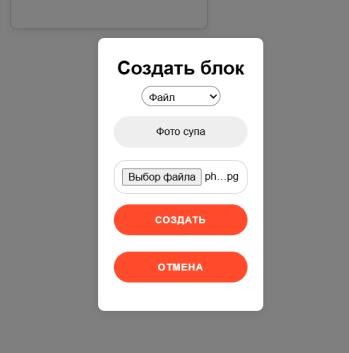
Для разработки дизайна интерфейса мы использовали Figma. Основные принципы дизайна:

* + - Минимализм и интуитивность.
    - Совместимость с различными устройствами.
    - Быстрый доступ к основным функциям.

Прототип включает:

* + - Основную страницу с заголовком.
    - Экран с перечнем вакансий и быстрым поиском по ним.



* + - Страницу с результатами поиска.
  1. Требования к программируемой системе Функциональные требования:

1. Правильная авторизация приложения в стороннем API.
2. Взаимодействия с .env файлом для избежания переавторизаций приложения.
3. Изолирование авторизованных запросов к API от клиента.
4. Автоматизация регулярных запросов.
5. Работа с JSON результатами: подсчет упоминаний и сортировка.
6. Буферизация обработанных данных.

Нефункциональные требования:

1. Высокая производительность и масштабируемость.
2. Удобство использования интерфейса.
3. Защита данных приложения от компрометирования.
   1. Метод и технология тестирования программируемой системы

Для тестирования запросов использовался Postman. Frontend тестировался вручную без применения автоматизированных тестов.

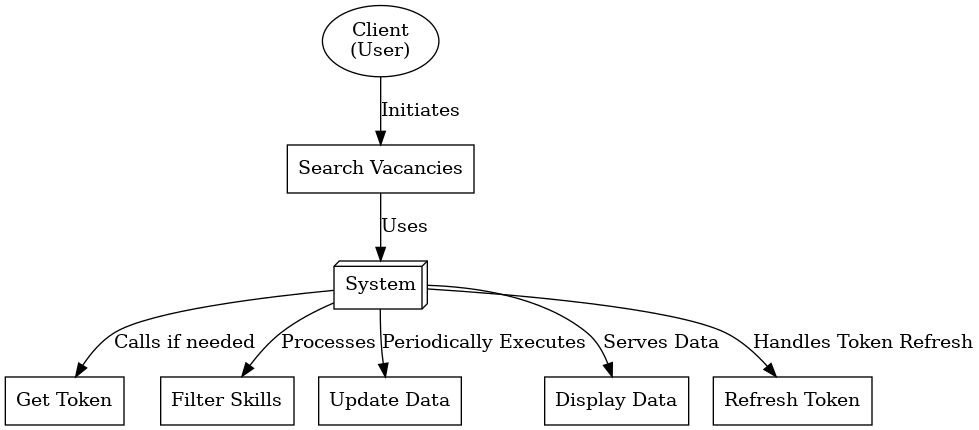
Мы использовали модульные тесты для проверки отдельных компонентов системы. Для функционального тестирования применялись автоматизированные сценарии, разработанные с использованием Pytest. Тестирование охватило следующие аспекты:

* + - Корректность обработки пользовательских запросов.
    - Надежность сохранения данных.
    - Производительность под нагрузкой.

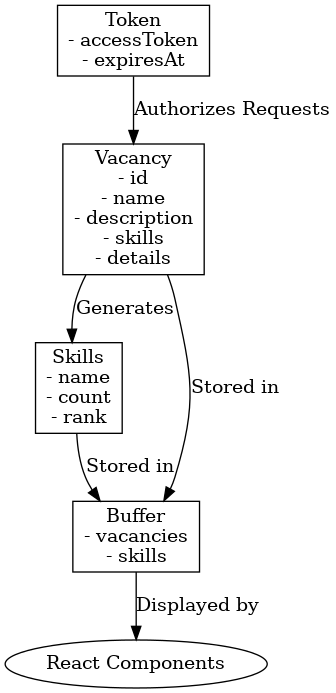
# Глава 2. Технологическая часть

2.1. Диаграммы

Use-Case диграмма:



ER-диаграмма



# Глава 3. Практическая часть

* 1. Описание программируемой системы

Система реализована с использованием Next.js для backend и React для frontend. Архитектура проекта следует Onion-архитектуре, обеспечивая четкое разделение логики.

* 1. Реализация требований к системе

Функциональные требования выполнены не полностью, присутствует имитация серверной части, но данные запрашивает и обрабатывает клиентская часть. Авторизация к hh API осуществлена, полученные токены скрываются от пользователя и хранятся в окружении. У разработчиков присутствует индивидуальный файл окружения, что ограничивает их влияние на основные методы приложения. Данные получаемые из ответов правильно обрабатываются и сортируются, возможно их сохранение для будущей быстрой загрузки результатов.

* 1. Функциональное тестирование программного продукта

Тестирование показало высокую стабильность работы системы, время работы отличается от ожиданий, сбоев и критических ошибок не выявлено. Запросы правильно формируются в зависимости от ввода пользователя, ответы правильно обрабатываются, сортируются и выводятся.

* 1. Инструкция по эксплуатации

Для начала работы с системой пользователь должен:

* + 1. Выбрать из списка интересующую вакансию или найти через поиск.
    2. После загрузки пользователь будет переведен на страницу с самыми востребованными технологиями и количеством их упоминания в вакансиях.

# Заключение

Практическая работа позволила нашей команде:

1. Овладеть навыками командной разработки.
2. Применить современные технологии и подходы в создании web-приложений.
3. Создать полноценный программный продукт, соответствующий заявленным требованиям.

# Список используемых источников

1. **Agile Manifesto [Электронный ресурс].** – URL: <https://agilemanifesto.org/> (дата обращения: 20.12.2024).
2. **Яндекс практикум. Методология Kanban [Электронный ресурс].** – URL: <https://practicum.yandex.ru/blog/metodologiya-kanban/> (дата обращения: 20.12.2024).
3. **hhAPI Documentation [Электронный ресурс].** – URL: <https://dev.hh.ru/> (дата обращения: 20.12.2024).
4. **React Documentation [Электронный ресурс].** – URL: <https://react.dev/> (дата обращения: 20.12.2024).
5. **Tailwind Documentation [Электронный ресурс].** – URL: <https://tailwindcss.com/docs/installation> (дата обращения: 20.12.2024).
6. **SQLAlchemy Documentation [Электронный ресурс].** – URL:
7. **Mozilla Developer Network (MDN). HTML, CSS, JavaScript и другие веб-технологии [Электронный ресурс].** – URL: <https://developer.mozilla.org/> (дата обращения: 20.12.2024).
8. **GitHub Actions Documentation [Электронный ресурс].** – URL: <https://docs.github.com/en/actions> (дата обращения: 20.12.2024).
9. **Docker Documentation [Электронный ресурс].** – URL: <https://docs.docker.com/> (дата обращения: 20.12.2024).
10. **Redis Documentation [Электронный ресурс].** – URL: <https://redis.io/documentation> (дата обращения: 20.12.2024).
11. **Node.js Documentation [Электронный ресурс].** – URL: <https://nodejs.org/en/docs/> (дата обращения: 20.12.2024).
12. **Express.js Documentation [Электронный ресурс].** – URL: <https://expressjs.com/> (дата обращения: 20.12.2024).
13. **Nginx Documentation [Электронный ресурс].** – URL: <https://nginx.org/en/docs/> (дата обращения: 20.12.2024).
14. **Next.js Documentation [Электронный ресурс].** – URL: <https://nextjs.org/docs> (дата обращения: 20.12.2024).
15. **MongoDB Documentation [Электронный ресурс].** – URL: <https://www.mongodb.com/docs/> (дата обращения: 20.12.2024).
16. **PostgreSQL Documentation [Электронный ресурс].** – URL: <https://www.postgresql.org/docs/> (дата обращения: 20.12.2024).
17. **Kubernetes Documentation [Электронный ресурс].** – URL: <https://kubernetes.io/docs/> (дата обращения: 20.12.2024).
18. **WebSocket Documentation [Электронный ресурс].** – URL: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API> (дата обращения: 20.12.2024).

# Приложения

Приложение A – apiOauth.js

let accessToken = *process*.env.REACT\_APP\_ACCESS\_TOKEN;  
let tokenExpiresAt = null;  
  
const getToken = async () => {  
 const currentTime = *Math*.floor(Date.now() / 1000);  
  
 if (!accessToken || (tokenExpiresAt && currentTime >= tokenExpiresAt)) {  
 try {  
 *console*.log("Запрашиваем новый токен...");  
  
 const response = await fetch("https://hh.ru/oauth/token", {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/x-www-form-urlencoded",  
 },  
 body: new *URLSearchParams*({  
 grant\_type: "client\_credentials",  
 client\_id: *process*.env.REACT\_APP\_CLIENT\_ID,  
  
 client\_secret: *process*.env.REACT\_APP\_CLIENT\_SECRET,  
 }),  
 });  
  
 if (!response.ok) {  
 const errorData = await response.json();  
 throw new Error(`Ошибка получения токена: ${errorData.error\_description || "Неизвестная ошибка"}`);  
 }  
  
 const data = await response.json();  
  
  
 accessToken = data.access\_token;  
 tokenExpiresAt = currentTime + data.expires\_in;  
  
 *console*.log("Токен успешно получен:", accessToken);  
 } catch (error) {  
 *console*.error("Ошибка получения токена:", error.message);  
 throw new Error("Ошибка авторизации HH API");  
 }  
 } else if ((tokenExpiresAt - currentTime) <= 360) {  
 // Если срок жизни токена менее 6 минут, не обновляем его  
 throw new Error("Вы пытаетесь получить токен слишком рано");  
 } else {  
 *console*.log("Используем кэшированный токен");  
 }  
  
 return accessToken;  
};  
  
*module*.exports = { getToken, accessToken };

Приложение B – requests.js

import *axios* from 'axios';  
import wordFilter from "../searchEngine/wordFilter";  
const { accessToken } = require("./apiOauth.js");  
  
const delay = (ms) => new Promise(resolve => setTimeout(resolve, ms));  
// Функция для получения ключевых навыков из вакансий  
const getKeySkills = async (query) => {  
 let allVacancies = [];  
 let page = 1;  
 const per\_page = 50;  
  
 const url = "https://api.hh.ru/vacancies";  
  
 try {  
 // Запрос на все страницы вакансий  
 while (true) {  
 const response = await axios.get(url, {  
 headers: {  
 Authorization: `Bearer ${accessToken}`,  
 },  
 params: {  
 text: query,  
 per\_page: per\_page,  
 page: page,  
 },  
 });  
  
 const vacancies = response.data.items;  
 if (vacancies.length === 0) break;  
  
 allVacancies.push(...vacancies);  
 page++;  
 if ((page+1) > response.data.pages) break;  
  
 await delay(3000);  
 }  
  
 const allSkills = [];  
  
 // Подробный запрос по каждой вакансии  
 const enrichedVacancies = [];  
 for (const vacancy of allVacancies) {  
 try {  
 const vacancyResponse = await axios.get(  
 `https://api.hh.ru/vacancies/${vacancy.id}`,  
 {  
 headers: {  
 Authorization: `Bearer ${accessToken}`,  
 },  
 }  
 );  
  
 const vacancyData = vacancyResponse.data;  
 const skills =  
 vacancyData.key\_skills?.map((skill) => skill.name).join(", ") ||  
 "Не указано";  
  
 enrichedVacancies.push({  
 ...vacancy,  
 skills,  
 });  
 allSkills.push(...vacancyData.key\_skills.map(skill => skill.name));  
 // Нужный нам массив для вывода на фронт  
 } catch (err) {  
 *console*.error(  
 `Ошибка при запросе деталей вакансии ${vacancy.id}:`,  
 err.message  
 );  
 continue;  
 }  
 }  
  
 // Фильтруем ключевые навыки  
 const filteredSkills = allSkills.filter((skill) =>  
 wordFilter(skill)  
 );  
  
 // Подсчитываем количество упоминаний каждого навыка  
 const skillCounts = filteredSkills.reduce((acc, skill) => {  
 acc[skill] = (acc[skill] || 0) + 1;  
 return acc;  
 }, {});  
  
 // Сортируем по убыванию  
 const sortedSkills = *Object*.entries(skillCounts)  
 .sort((a, b) => b[1] - a[1])  
 .map(([skill, count], index) => ({ rank: index + 1, skill, count })

.slice(0, 100));

return {  
 enrichedVacancies: allVacancies,  
 skillCounts: sortedSkills,  
 };  
 } catch (error) {  
 *console*.error("Ошибка при получении ключевых навыков:", error.message);  
 throw new Error("Ошибка при получении ключевых навыков");  
 }  
};  
  
export { getKeySkills };

export const getTopSkills = async (query) => {  
 const { skillCounts } = await getKeySkills(query);  
 return skillCounts;  
};

Приложение C – scheduler.js

import fs from "fs/promises";  
import {getKeySkills, getTopSkills} from "./requests.js";  
  
const updateSkillsData = async () => {  
 try {  
 const query = "Frontend Developer"; //заменить в будущем  
 const { skillCounts } = await getKeySkills(query);  
  
 //объект для сохранения  
 const data = {  
 updated\_at: new *Date*().toISOString(),  
 skills: getTopSkills  
 };  
  
 // Сохраняем в JSON-файл  
 await fs.writeFile("skills.json", *JSON*.stringify(data, null, 2));  
 *console*.log("Данные успешно обновлены");  
 } catch (error) {  
 *console*.error("Ошибка при обновлении данных:", error.message);  
 }  
};  
  
// Запуск скрипта  
updateSkillsData();

Приложение D – server.js

// server  
const express = require("express");  
const { getVacancies } = require("./requests.js");  
const cors = require("cors");  
  
const app = express();  
const port = 8000;  
  
app.use(cors({  
 origin: 'http://localhost:3000',  
 methods: ['GET', 'POST'],  
 credentials: true,  
}));  
// Эндпоинт для получения вакансий  
app.get("/api/vacancies", async (req, res) => {  
 const query = req.query.query;  
  
 if (!query) {  
 return res.status(400).json({ error: "Параметр query не найден" });  
 }  
  
 try {  
 const vacancies = await getVacancies(query); // Получаем вакансии с использованием функции  
 res.json(vacancies); // Отправляем вакансии клиенту  
 } catch (error) {  
 *console*.error("Ошибка при получении вакансий:", error);  
 res.status(500).json({ error: "Ошибка при получении вакансий" });  
 }  
});  
  
app.listen(port, () => {  
 *console*.log(`Server is running on http://localhost:${port}`);  
});

Приложение E – wordfilter.js

const excludedWords = [  
 "1", "2", "3", "4", "5", "6", "7", "8", "9", "etc",  
 "it", "00", "10", "100", "quot", "11", "12", "13", "14", "15", "16", "17", "18",  
 "and", "the", "to", "end", "39", "ru", "of", "000", "you", "in", "skills", "back", "with", "for",  
 "we", "on",  
 "19", "e", "er", "30", "20", "0", "a", "o", "50",  
 "be", "our", "will", "is", "your", "as", "that",  
 "including", "an", "new", "are", "at", "by",  
 "other", "have", "all", "about", "us", "ozon", "music",  
 "s7", "on", "1000", "or", "off", "70", "200",  
 "based", "ready", "60", "80", "90", "400", "what",  
 "from", "40", "war", "24", "skyeng", "tinkoff", "09",  
 "2015", "wildberries", "08", "300", "25", "45", "need",  
 "do", "28", "500", "2017", "112", "103",  
 "122", "2008", "232", "485", "3000", "ll", "2022",  
 "400"  
];  
  
const wordFilter = (word) => {  
 // Сравниваем слово с исключёнными словами  
 return !excludedWords.includes(word.toLowerCase());  
};  
  
export default wordFilter;