|  |  |  |
| --- | --- | --- |
| Защищено:  Нардид А.Н.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |  | Демонстрация:  Нардид А.Н.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Отчет по домашнему заданию по курсу**

**Парадигмы и конструкции языков программирования**

#### Тема работы: "Конвертация русских слов в катакану"

11

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-53Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Ахмеров Д.И. | "5" декабря 2024 г. |

Москва, МГТУ – 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Описание заданий**

Разработать веб-приложение на стеке технологий React + Typescript, которое конвертирует русские слова в катакану

**Текст программ**

**main.tsx**

import { StrictMode } from 'react'

import { createRoot } from 'react-dom/client'

import Converter from './components/Converter.tsx'

createRoot(document.getElementById('root')!).render(

<StrictMode>

<Converter />

</StrictMode>,

)

**Converter.tsx**

import React, { useState } from "react";

import { transform } from "../utils/transform";

const Converter: React.FC = () => {

const [input, setInput] = useState<string>("");

const [result, setResult] = useState<string>("");

const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {

setInput(event.target.value);

};

const handleConvert = () => {

try {

const [katakana, syllables] = transform(input);

setResult(`

Катакана: ${katakana.join("")}

Слоги: ${syllables.join("-")}

`);

} catch (error) {

setResult("Error occurred during transformation");

console.error(error);

}

};

return (

<div style={{ padding: "20px", fontFamily: "Arial, sans-serif" }}>

<h1>Конвертер</h1>

<input

type="text"

value={input}

onChange={handleChange}

placeholder="Введите слово..."

style={{

padding: "10px",

fontSize: "16px",

width: "300px",

marginBottom: "10px",

}}

/>

<br />

<button

onClick={handleConvert}

style={{

padding: "10px 20px",

fontSize: "16px",

cursor: "pointer",

backgroundColor: "#4CAF50",

color: "white",

border: "none",

borderRadius: "5px",

}}

>

Конвертировать

</button>

<div

style={{

marginTop: "20px",

whiteSpace: "pre-wrap",

fontSize: "14px",

color: "#333",

}}

>

{result}

</div>

</div>

);

};

export default Converter;

**katakanize.ts**

export function katakanize(

syllable: string,

index: number,

syllables: string[],

): string {

const commonPairs: Record<string, string> = {

'а': 'ア',

'и': 'イ',

'у': 'ウ',

'э': 'エ',

'о': 'オ',

'ка': 'カ',

'ки': 'キ',

'ку': 'ク',

'кэ': 'ケ',

'ко': 'コ',

'са': 'サ',

'щи': 'シ',

'си': 'シ',

'су': 'ス',

'сэ': 'セ',

'со': 'ソ',

'та': 'タ',

'чи': 'チ',

'цу': 'ツ',

'тэ': 'テ',

'то': 'ト',

'на': 'ナ',

'ни': 'ニ',

'ну': 'ヌ',

'нэ': 'ネ',

'но': 'ノ',

'ха': 'ハ',

'хи': 'ヒ',

'фу': 'フ',

'хэ': 'ヘ',

'хо': 'ホ',

'ма': 'マ',

'ми': 'ミ',

'му': 'ム',

'мэ': 'メ',

'мо': 'モ',

'я': 'ヤ',

'ю': 'ユ',

'ё': 'ヨ',

'ра': 'ラ',

'ри': 'リ',

'ру': 'ル',

'рэ': 'レ',

'ро': 'ロ',

'па': 'パ',

'пи': 'ピ',

'пу': 'プ',

'пэ': 'ペ',

'по': 'ポ',

'ба': 'バ',

'би': 'ビ',

'бу': 'ブ',

'бэ': 'ベ',

'бо': 'ボ',

'да': 'ダ',

'дзи': 'ヂ',

'дзу': 'ヅ',

'дэ': 'デ',

'до': 'ド',

'за': 'ザ',

'зи': 'ジ',

'зу': 'ズ',

'зэ': 'ゼ',

'зо': 'ゾ',

'га': 'ガ',

'ги': 'ギ',

'гу': 'グ',

'гэ': 'ゲ',

'го': 'ゴ',

'н': 'ン',

'м': 'ン',

'кя': 'キャ',

'кю': 'キュ',

'кё': 'キョ',

'ся': 'シャ',

'сю': 'シュ',

'сё': 'ショ',

'тя': 'チャ',

'тю': 'チュ',

'тё': 'チョ',

'ня': 'ニャ',

'ню': 'ニュ',

'нё': 'ニョ',

'хя': 'ヒャ',

'хю': 'ヒュ',

'хё': 'ヒョ',

'мя': 'ミャ',

'мю': 'ミュ',

'мё': 'ミョ',

'ря': 'リャ',

'рю': 'リュ',

'рё': 'リョ',

'пя': 'ピャ',

'пю': 'ピュ',

'пё': 'ピョ',

'бя': 'ビャ',

'бю': 'ビュ',

'бё': 'ビョ',

'дя': 'ヂャ',

'дю': 'ヂュ',

'дё': 'ヂョ',

'дзя': 'ジャ',

'дзю': 'ジュ',

'дзё': 'ジョ',

'гя': 'ギャ',

'гю': 'ギュ',

'гё': 'ギョ',

'+': 'ッ'

};

if (commonPairs[syllable]) {

return commonPairs[syllable];

}

return syllable;

}

**prepare\_doubles.ts**

const singlesConsonants: string[] = [

'б',

'в',

'г',

'д',

'з',

'й',

'к',

'м',

'н',

'п',

'р',

'с',

'т',

'ф',

'ц',

'ч',

'х',

'щ',

'ш'

];

const doubleConsonants: string[] = ['н', 'м'];

export function prepareDoubles(syllable: string, index: number, syllables: string[]): string {

if (

singlesConsonants.includes(syllable) &&

!doubleConsonants.includes(syllable)

) {

return '+';

}

return syllable;

}

**replace.ts**

export function replace(

syllable: string,

index: number,

syllables: string[]

): string {

console.log(`Processing syllable: ${syllable}, index: ${index}`);

if (singlesConsonants[syllable]) {

return singlesConsonants[syllable];

}

if (pairs[syllable]) {

const replacement = pairs[syllable];

const result = Array.isArray(replacement) ? replacement.join('') : replacement;

return result;

}

let result = syllable;

for (const searchChars in replacePairs) {

if (Object.prototype.hasOwnProperty.call(replacePairs, searchChars)) {

const regex = new RegExp(searchChars, 'g');

const replaced = result.replace(regex, replacePairs[searchChars]);

if (replaced !== result) {

result = replaced;

}

}

}

return result;

}

const pairs: Record<string, string | string[]> = {

'мы': ['му', 'и'],

'ху': 'фу',

'фу': 'фу',

'цы': 'цу',

'ть': 'чи',

'чи': 'чи',

'ча': 'тя',

'чу': 'тю',

'чо': 'тё',

'ти': 'чи',

'ту': 'цу',

'цу': 'цу',

'ди': 'дзи',

'ду': 'зу',

'жу': 'зю',

'жо': 'зё',

'жа': 'дзя',

'зя': 'дзя',

'зю': 'дзю',

'зё': 'дзё',

'жэ': 'зэ',

'ша': 'ся',

'ща': 'ся',

'шу': 'сю',

'щу': 'сю',

'щи': 'щи',

'ши': 'щи',

'си': 'щи',

'що': 'сё',

'шо': 'сё',

};

const replacePairs: Record<string, string> = {

'ь': 'и',

'щ': 'с',

'ш': 'с',

'йо': 'ё',

'йу': 'ю',

'йа': 'я',

'в': 'б',

'л': 'р',

'ы': 'у',

'е': 'э',

'ц': 'т',

'ч': 'т',

'ф': 'х',

'ж': 'з',

};

const singlesConsonants: Record<string, string> = {

'б': 'бу',

'в': 'бу',

'г': 'гу',

'д': 'до',

'ж': 'зу',

'з': 'зу',

'й': 'и',

'к': 'ку',

'л': 'ру',

'м': 'му',

'п': 'пу',

'р': 'ру',

'с': 'су',

'т': 'то',

'ф': 'фу',

'ц': 'цу',

'ч': 'чи',

'х': 'хо',

'ш': 'щи',

'щ': 'щи'

};

**syllable.ts**

export function syllable(word: string): string[] {

const vowels = ['а', 'е', 'ё', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я', 'ь'];

const consonants = [

'б', 'в', 'г', 'д', 'ж', 'з', 'й', 'к', 'л', 'м',

'н', 'п', 'р', 'с', 'т', 'ф', 'ц', 'ч', 'х', 'ш', 'щ'

];

const complexSyllables = ['дз'];

const special = ['ъ'];

const arr = word.toLowerCase().split('');

const syllables: string[] = [];

let currentSyllable: string[] = [];

function endSyllable() {

syllables.push(currentSyllable.join(''));

currentSyllable = [];

}

function addLetter(letter: string) {

currentSyllable.push(letter);

}

arr.forEach((letter) => {

if (vowels.includes(letter)) {

addLetter(letter);

endSyllable();

} else if (

consonants.includes(letter) &&

complexSyllables.includes(currentSyllable.join('') + letter)

) {

addLetter(letter);

} else if (

consonants.includes(letter) &&

currentSyllable.length !== 0

) {

if (

currentSyllable[0] === letter &&

syllables.length === 0

) {

return;

}

endSyllable();

addLetter(letter);

} else if (

consonants.includes(letter) &&

currentSyllable.length === 0

) {

addLetter(letter);

} else if (special.includes(letter)) {

endSyllable();

}

});

if (currentSyllable.length) {

endSyllable();

}

return syllables;

}

**transform.ts**

import {replace} from './replace';

import {syllable} from './syllable';

import {katakanize} from './katakanize';

import {prepareDoubles} from './prepare\_doubles';

export function transform(word: string): [string[], string[], string[]] {

const chain: string[] = [word];

// Разбиение на слоги

let syllables: string[] = syllable(word);

chain.push(syllables.join('-'));

// Трансформация под произношение

syllables = syllables.map((syl) => replace(syl, 0, syllables)) as string[];

syllables = flat(syllables);

chain.push(syllables.join('-'));

// Обработка для катаканизации

const prepared: string[] = syllables.map((syl) => prepareDoubles(syl, 0, syllables));

chain.push(prepared.join('-'));

// Катаканизация

const katakana: string[] = prepared.map((syl) => katakanize(syl, 0, syllables));

chain.push(katakana.join(''));

return [katakana, syllables, chain];

}

function flat(syllables: (string | string[])[]): string[] {

const result: string[] = [];

syllables.forEach((syllable) => {

if (typeof syllable === 'string') {

result.push(syllable);

} else {

syllable.forEach((newSyllable) => {

result.push(newSyllable);

});

}

});

return result;

}

**Экранная форма с примером выполнения программы**

