Отчет по лабораторной работе № 7 по курсу
"Разработка Интернет-Приложений"

Выполнил:
Студент группы
ИУ5-55Б
Хижняков Вадим Максимович

Москва, МГТУ – 2021

Задание:
На основе методических указаний разработайте пользовательский интерфейс для работы с REST API. Используйте REST API, разработанный Вами в предыдущей лабораторной работе.

В качестве клиента на js будет представлено приложение по вычислению кода Хэмминга.

С помощью библиотеки express создан сервер на typescript, отдающий статические файлы, а также выполняющий вычисления.

Текст программы:

Модуль для вычислений

```
export const bitsCount = (a: number) => Math.ceil(Math.log2(a + 1));

export const div = (a: number, b: number) => (a - (a % b)) / b;

const bitOnesCount = (a: number) => {
    let onesCount = 0;
    while (a !== 0) {
        onesCount += a % 2;
        a = div(a, 2);
    }
    return onesCount;
};

export const getCountOfControlBits = (binLength: number) => {
    let k = bitsCount(binLength);
    while (k !== Math.ceil(Math.log2(binLength + +(binLength !== 2) + k))) {
        k++;
    }
    return k;
};

export const isEqualArrays = <T extends unknown>(a: T[], b: T[]) =>
    a.length === b.length && a.every((_, i) => a[i] === b[i]);

export const generateErrors = (codedWord: boolean[]) =>
```

```
    Array.from({ length: 2 ** codedWord.length }, (_, index) =>
index).reduce<Record<number, number[]>>(
        (accum, num) => {
            const arr = (accum[bitOnesCount(num)] ??= []);
            arr.push(num);
            return accum;
        },
        {}
    );

export const applyError = (_word: boolean[], error: number) => {
    const word = [..._word];
    for (let i = word.length - 1; error !== 0; i--) {
        if (error % 2 !== 0) {
            word[i] = !word[i];
        }
        error = div(error, 2);
    }
    return word;
};

export const multiplyRows = (message: boolean[], rowNumber: number): boolean => {
    const step = 2 ** rowNumber;

    let controlBit = false;

    for (let i = 0, j = div(1, step); i < message.length; i++, j = div(i + 1, step)) {
        if (j % 2 !== 0) {
            controlBit = controlBit !== message[i];
        }
    }

    return controlBit;
};

export const hammingCode = (_message: boolean[]) => {
    const message = [..._message];
    const countOfControlBits = getCountOfControlBits(message.length);

    for (let i = 0; i < countOfControlBits; i++) {
        message.splice(2 ** i - 1, 0, false);
    }

    for (let i = 0; i < countOfControlBits; i++) {
        message[2 ** i - 1] = multiplyRows(message, i);
    }

    return message;
};

export const hammingDecode = (_message: boolean[]): [decoded: boolean[], error: boolean,
syndrom: boolean[]] => {
    const message = [..._message];
    const countOfControlBits = Math.ceil(Math.log2(message.length));
```

```
    const syndrom = [];
    let errorIndex = -1;
    for (let indexOfControlBit = 0; indexOfControlBit < countOfControlBits;
indexOfControlBit++) {
        const isErrorBit = multiplyRows(message, indexOfControlBit);
        if (isErrorBit) {
            errorIndex += 2 ** indexOfControlBit;
        }
        syndrom.unshift(isErrorBit);
    }

    if (errorIndex !== -1) {
        message[errorIndex] = !message[errorIndex];
    }

    for (let i = countOfControlBits - 1; i >= 0; i--) {
        message.splice(2 ** i - 1, 1);
    }
    return [message, errorIndex !== -1, syndrom];
};

export const createBooleanMesssage = (msg: string) => msg.split("").map((char) => char ===
"1");

export const parseBooleanMesssage = (msg: boolean[]) => msg.map((cur) => (cur ? "1" :
"0")).join("");

export const formatCode = (num: number, length: number) =>
num.toString(2).padStart(length, "0");
```

Сервер

```
import express, { Request, Response } from "express";
import path from "path";
import morgan from "morgan";
import bp from "body-parser";

import {
    hammingCode,
    hammingDecode,
    generateErrors,
    createBooleanMesssage,
    parseBooleanMesssage,
    applyError,
    formatCode,
    isEqualArrays,
} from "./lib";

const port = process.env.PORT || 5000;

type HammingReq = {
    word?: string;
```

```
};

type HammingError = {
   error: string;
   syndrom: string;
   errorWord: string;
   decodedWord: string;
};

type ErrorMultiplicity = {
   multiplicity: string;
   errors: HammingError[];
   detectedCount: number;
   fixedCount: number;
};

type HammingRes = {
   word: string;
   codedWord: string;
   errors: ErrorMultiplicity[];
};

const app = express();
app.use(morgan("dev"));
app.use(bp.json());
app.use(express.static(path.join(__dirname, "static")));

app.get("/", (_, res) => res.sendFile(path.join(__dirname, "static", "index.html")));

app.post("/hamming", (req: Request<{}, {}, HammingReq>, res: Response<HammingRes>) => {
   const word = createBooleanMesssage(req.body?.word ?? "01010101111");

   const codedWord = hammingCode(word);

   const errors = Object.entries(generateErrors(codedWord)).reduce<ErrorMultiplicity[]>(
       (accum, [multiplicity, errors]) => {
           let detectedCount = 0;
           let fixedCount = 0;
           const multiplicityErrors = errors.map<HammingError>((error) => {
               const codedWithError = applyError(codedWord, error);
               const [decoded, detected, syndrom] = hammingDecode(codedWithError);
               detectedCount += +detected;
               fixedCount += +isEqualArrays(decoded, word);
               return {
                   error: formatCode(error, codedWithError.length),
                   errorWord: parseBooleanMesssage(codedWithError),
                   decodedWord: parseBooleanMesssage(decoded),
                   syndrom: parseBooleanMesssage(syndrom),
               };
           });
           accum[+multiplicity] = {
               multiplicity,
               errors: multiplicityErrors,
               detectedCount,
               fixedCount,
```

```
            };
            return accum;
        },
        []
    );

    return res.send({
        word: parseBooleanMesssage(word),
        codedWord: parseBooleanMesssage(codedWord),
        errors,
    });
});
app.listen(port, () => console.log(`server is running in http://localhost:${port}`));
```

Клиентская часть:

index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <link rel="stylesheet" href="./style.css" />
        <title>SETI DZ</title>
    </head>
    <body>
        <div id="abilities"></div>
        <div class="form">
            <input type="number" id="word" />
            <button type="button" id="send">Посчитать</button>
        </div>
        <div id="result"></div>
        <div id="loader" class="loader">
            <img src="./spinner.svg" alt="Loading" />
        </div>
    </body>
    <script src="script.js" lang="js"></script>
</html>
```

script.js

```
/**
*
* @param {string[]} array
* @returns
*/
```

```
const createTableRow = (...array) =>
`<tr><td><pre>${array.join("</pre></td><td><pre>")}</pre></td></tr>`;

/**
*
* @param {{ error: string, syndrom: string, errorWord: string, decodedWord: string}[]}
errors
* @param {string} multiplicity
* @returns
*/
const createErrorTable = (errors, multiplicity) => `
<h3>Кратность ошибки - ${multiplicity}</h3>
<table>
   <tr>
       <th>Ошибка</th>
       <th>Синдром</th>
       <th>Закодированное слово</th>
       <th>Раскодированное слово</th>
   </tr>
   ${errors
       .slice(0, 100)
       .map(({ error, errorWord, decodedWord, syndrom }) => createTableRow(error, syndrom,
errorWord, decodedWord))
       .join("")}
</table>`;

/**
*
* @param {{ errors: { multiplicity: string, detectedCount: number, fixedCount: number,
errors: { error: string, syndrom: string, errorWord: string, decodedWord: string}[]}[]}}
result
* @returns
*/
const createAbilitiesTable = ({ errors }) =>
   `
<table class="columns">
   <tr>
       <td>Кратность ошибки</td>
       <td>Обнаруживающая способность, %</td>
       <td>Корректирующая способность, %</td>
       <td>Обнаруженные ошибки</td>
       <td>Скорректированные ошибки</td>
       <td>Скорректированные к обнаруженным</td>
       <td>Общее число ошибок</td>
   </tr>
   ${errors
       .map(
           ({ multiplicity, detectedCount, fixedCount, errors }) => `
   <tr>
       <td>${multiplicity}</td>
       <td>${((detectedCount * 100) / errors.length).toFixed(2).replace(/(\.0)?0$/,
"")}%</td>
       <td>${((fixedCount * 100) / errors.length).toFixed(2).replace(/(\.0)?0$/, "")}%</td>
       <td>${detectedCount}</td>
       <td>${fixedCount}</td>
```

```
        <td>${fixedCount / detectedCount}</td>
        <td>${errors.length}</td>
    </tr>
`
        )
        .join("")}
</table>`;

/**
*
* @param {string} word
* @param {string} codedWord
* @returns
*/
const createHammingInfo = (word, codedWord) => `
<table class="info">
    <tr>
        <th>Слово</th>
        <th>Закодированное слово</th>
    </tr>
    <tr>
        <td><pre>${word}</pre></td>
        <td><pre>${codedWord}</pre></td>
    </tr>
</table>`;

/**
*
* @param {string} word
* @returns {Promise<{ word: string, codedWord: string, errors: { multiplicity: string,
detectedCount: number, fixedCount: number, errors: { error: string, syndrom: string,
errorWord: string, decodedWord: string}[]}[]}>}
*/
const sendReq = async (word) =>
    await (
        await fetch("/hamming", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ word }),
        })
    ).json();

/**
*
* @param {{ word: string, codedWord: string, errors: { multiplicity: string,
detectedCount: number, fixedCount: number, errors: { error: string, syndrom: string,
errorWord: string, decodedWord: string}[]}[]}} result
* @returns
*/
const renderResult = ({ codedWord, word, errors }) => {
    document.getElementById("loader").classList.remove("visible");
    document.getElementById("result").innerHTML =
        createHammingInfo(word, codedWord) +
```

```
        errors.map(({ errors, multiplicity }) => createErrorTable(errors,
multiplicity)).join("");
    document.getElementById("abilities").innerHTML = createAbilitiesTable({ errors });
};

/**
*
* @param {string} value
*/
const attachResult = (value) => {
    document.getElementById("loader").classList.add("visible");
    sendReq(value).then(renderResult);
};

const main = async () => {
    const result = await sendReq();
    renderResult(result);
    const input = document.getElementById("word");
    document.getElementById("send").addEventListener("click", () =>
attachResult(input.value));
    input.addEventListener("input", (e) => (e.target.value =
e.target.value.replace(/[^01]/g, "")));
    input.addEventListener("keydown", (e) => (e.key.toLowerCase() === "enter" ?
attachResult(input.value) : true));
};

window.onload = main;
```

Результат работы программы:

Кратность ошибки	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Обнаруживающая способность, %	0%	100%	100%	92.31%	92.31%	94.41%	94.41%	93.24%	93.24%	94.41%	94.41%	92.31%	92.31%	100%	100%	0%
Корректирующая способность, %	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Обнаруженные ошибки	0	15	105	420	1260	2835	4725	6000	6000	4725	2835	1260	420	105	15	0
Скорректированные ошибки	1	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Скорректированные к обнаруженным	Infinity	1	0	0	0	0	0	0	0	0	0	0	0	0	0	NaN
Общее число ошибок	1	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1

Посчитать

Слово	Закодированное слово
01010101111	000010110101111

Кратность ошибки - 0

Ошибка	Синдром	Закодированное слово	Раскодированное слово
000000000000000	0000	000010110101111	01010101111

Кратность ошибки - 1

Ошибка	Синдром	Закодированное слово	Раскодированное слово
000000000000001	1111	000010110101110	01010101111
000000000000010	1110	000010110101101	01010101111
000000000000100	1101	000010110101011	01010101111
000000000001000	1100	000010110100111	01010101111
000000000010000	1011	000010110111111	01010101111
000000000100000	1010	000010110001111	01010101111
000000001000000	1001	000010111101111	01010101111
000000010000000	1000	000010100101111	01010101111

Кратность ошибки	0	1	2	3	4	5	6
Обнаруживающая способность, %	0%	100%	100%	80%	80%	100%	100%
Корректирующая способность, %	100%	100%	0%	0%	0%	0%	0%
Обнаруженные ошибки	0	6	15	16	12	6	1
Скорректированные ошибки	1	6	0	0	0	0	0
Скорректированные к обнаруженным	Infinity	1	0	0	0	0	0
Общее число ошибок	1	6	15	20	15	6	1

111 Посчитать

Слово	Закодированное слово
111	001011

Кратность ошибки - 0

Ошибка	Синдром	Закодированное слово	Раскодированное слово
000000	000	001011	111

Кратность ошибки - 1

Ошибка	Синдром	Закодированное слово	Раскодированное слово
000001	110	001010	111
000010	101	001001	111
000100	100	001111	111
001000	011	000011	111
010000	010	011011	111
100000	001	101011	111

Кратность ошибки - 2