|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа №4**

**по дисциплине «Архитектура ЭВМ»**

|  |  |
| --- | --- |
| **Студент:**Блохин Д.М.  **Группа:** ИУ7-54Б  **Преподаватель:** Попов А.Ю.  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

### **Цель работы**

### Изучение межсерверного взаимодействия и дочерних процессов.

### Изучение логических предикатов в концепции языка Prolog.

### **Подзадача 7**

### **Задание 1**

Создать сервер А. На стороне сервера хранится файл с содержимым в формате JSON. При получении запроса на /insert/record идёт добавление записи в файл. При получении запроса на /select/record идёт получение записи из файла. Каждая запись хранит информацию о машине (название и стоимость).

Создать сервер Б. На стороне сервера хранится файл с содержимым в формате JSON. Каждая запись в файле хранит информацию о складе и массиве машин, находящихся на данном складе. То есть каждая запись хранит в себе название склада (строку) и массив названий машин (массив строк). При получении запроса на /insert/record идёт добавление записи в файл. При получении запроса на /select/record идёт получение записи из файла.

Создать сервер C. Сервер выдаёт пользователю страницы с формами для ввода информации. При этом сервер взаимодействует с серверами А и Б. Реализовать для пользователя функции:

* создание нового типа машины;
* получение информации о стоимости машины по её типу;
* создание нового склада с находящимися в нём машинами;
* получение информации о машинах на складе по названию склада**.**

Реализовать удобный для пользователя интерфейс взаимодействия с системой (использовать поля ввода и кнопки).

**Реализация**

**A.js**

"use strict";

const express = require('express');

const fs = require('fs');

const app = express();

const port = 5001;

app.listen(port);

console.log(`Server on port ${port}`);

const workFile = \_\_dirname + '/vehicles.json';

const workType = "Vehicle";

if (!fs.existsSync(workFile)) {

fs.writeFileSync(workFile, "[]")

}

function loadBody(request, callback) {

let body = [];

request.on('data', (chunk) => {

body.push(chunk);

}).on('end', () => {

body = Buffer.concat(body).toString();

callback(body);

});

}

app.use(function(req, res, next) {

res.header('Cache-Control', 'no-cache, no-store, must-revalidate');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');

res.header('Access-Control-Allow-Origin', '\*');

next();

});

app.post('/insert/record', function(request, response) {

loadBody(request, function(body) {

const obj = JSON.parse(body);

const arrStr = fs.readFileSync(workFile);

const arr = JSON.parse(arrStr);

for (const elem of arr) {

if (elem.name === obj.name) {

response.end(JSON.stringify({result: workType + ' already exists'}));

return;

}

}

arr.push(obj);

fs.writeFileSync(workFile, JSON.stringify(arr));

response.end(JSON.stringify({result: workType + ' added'}));

});

});

app.get('/select/record', function(request, response) {

const name = request.query.name

const arrStr = fs.readFileSync(workFile)

const arr = JSON.parse(arrStr);

for (const elem of arr) {

if (elem.name === name) {

return response.end(JSON.stringify(elem));

}

}

return response.end(JSON.stringify({result: workType + ' not found'}));

});

**B.js**

"use strict";

const express = require('express');

const fs = require('fs');

const app = express();

const port = 5002;

app.listen(port);

console.log(`Server on port ${port}`);

const workFile = \_\_dirname + '/warehouses.json';

const workType = "Warehouse";

if (!fs.existsSync(workFile)) {

fs.writeFileSync(workFile, "[]")

}

function loadBody(request, callback) {

let body = [];

request.on('data', (chunk) => {

body.push(chunk);

}).on('end', () => {

body = Buffer.concat(body).toString();

callback(body);

});

}

app.use(function(req, res, next) {

res.header('Cache-Control', 'no-cache, no-store, must-revalidate');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');

res.header('Access-Control-Allow-Origin', '\*');

next();

});

app.post('/insert/record', function(request, response) {

loadBody(request, function(body) {

const obj = JSON.parse(body);

const arrStr = fs.readFileSync(workFile);

const arr = JSON.parse(arrStr);

for (const elem of arr) {

if (elem.name === obj.name) {

response.end(JSON.stringify({result: workType + ' already exists'}));

return;

}

}

arr.push(obj);

fs.writeFileSync(workFile, JSON.stringify(arr));

response.end(JSON.stringify({result: workType + ' added'}));

});

});

app.get('/select/record', function(request, response) {

const name = request.query.name

const arrStr = fs.readFileSync(workFile)

const arr = JSON.parse(arrStr);

for (const elem of arr) {

if (elem.name === name) {

return response.end(JSON.stringify(elem));

}

}

return response.end(JSON.stringify({result: workType + ' not found'}));

});

**C.js**

"use strict";

const express = require('express');

const request = require("request");

const app = express();

const port = 5003;

app.listen(port);

console.log(`Server on port ${port}`);

function loadBody(request, callback) {

let body = [];

request.on('data', (chunk) => {

body.push(chunk);

}).on('end', () => {

body = Buffer.concat(body).toString();

callback(body);

});

}

app.set('view engine', 'hbs');

app.use(function(req, res, next) {

res.header('Cache-Control', 'no-cache, no-store, must-revalidate');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');

res.header('Access-Control-Allow-Origin', '\*');

next();

});

function sendPost(url, body, callback) {

const headers = {};

headers["Cache-Control"] = "no-cache, no-store, must-revalidate";

headers["Connection"] = "close";

request.post({

url: url,

body: body,

headers: headers,

}, function (error, response, body) {

if (error) {

callback(null);

} else {

callback(body);

}

});

}

function sendGet(url, callback) {

const headers = {};

headers['Cache-Control'] = 'no-cache, no-store, must-revalidate';

headers['Connection'] = 'close';

request.get({

url: url,

headers: headers,

}, function (error, response, body) {

if (error) {

callback(null);

} else {

callback(body);

}

});

}

app.get('/vehicle/getForm', function (request, response) {

return response.render('vehicle\_get.hbs', {});

});

app.get('/vehicle/get', function (request, response) {

const name = encodeURIComponent(request.query.name);

sendGet(`http://localhost:5001/select/record?name=${name}`, function(answerString) {

const answerObject = JSON.parse(answerString);

if (answerObject.result) {

return response.render('vehicle\_get.hbs', {"result": answerObject.result});

}

else {

response.render('vehicle.hbs', answerObject);

}

});

});

app.get('/warehouse/getForm', function (request, response) {

return response.render('warehouse\_get.hbs');

});

app.get('/warehouse/get', function (request, response) {

const name = encodeURIComponent(request.query.name);

sendGet(`http://localhost:5002/select/record?name=${name}`, function(answerString) {

const answerObject = JSON.parse(answerString);

if (answerObject.result) {

return response.render('warehouse\_get.hbs', {"result": answerObject.result});

}

else {

response.render('warehouse.hbs', answerObject);

}

});

});

app.get('/vehicle/addForm', function (request, response) {

return response.render('vehicle\_add.hbs');

});

app.get('/vehicle/add', function (request, response) {

const name = request.query.name;

const price = request.query.price;

sendPost(`http://localhost:5001/insert/record`, JSON.stringify({"name":name, "price":price}), function(answerString) {

const answerObject = JSON.parse(answerString);

const answer = answerObject.result;

response.end(answer);

});

});

app.get('/warehouse/addForm', function (request, response) {

return response.render('warehouse\_add.hbs');

});

app.get('/warehouse/add', function (request, response) {

const name = request.query.name;

const vehicles = [];

for (const [key, value] of Object.entries(request.query)) {

if (key[0] === 'v') {

vehicles.push(value);

}

}

const whObj = {"name":name, "vehicles":vehicles}

sendPost(`http://localhost:5002/insert/record`, JSON.stringify(whObj), function(answerString) {

const answerObject = JSON.parse(answerString);

const answer = answerObject.result;

response.end(answer);

});

});

app.get('/', function(request, response) {

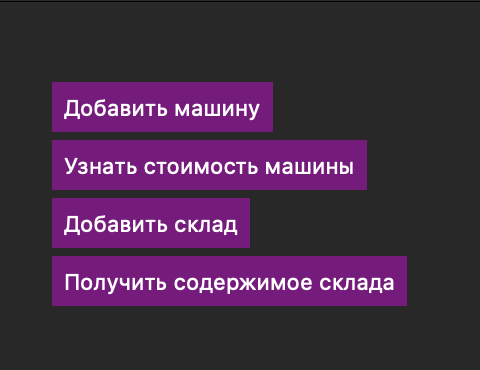
return response.redirect('/page.html');

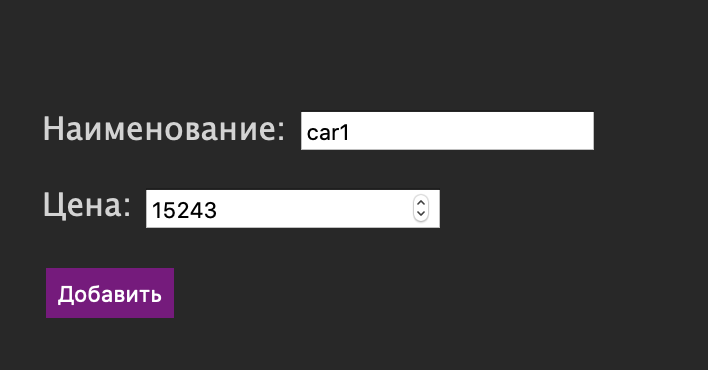
});

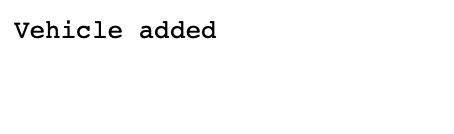
const way = \_\_dirname + '/static';

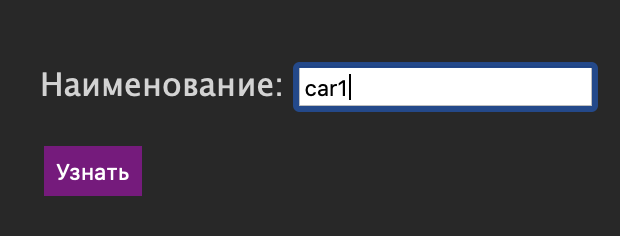
app.use(express.static(way));

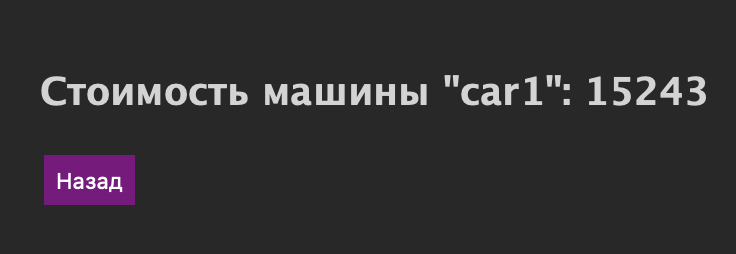
**Проверка работоспособности**

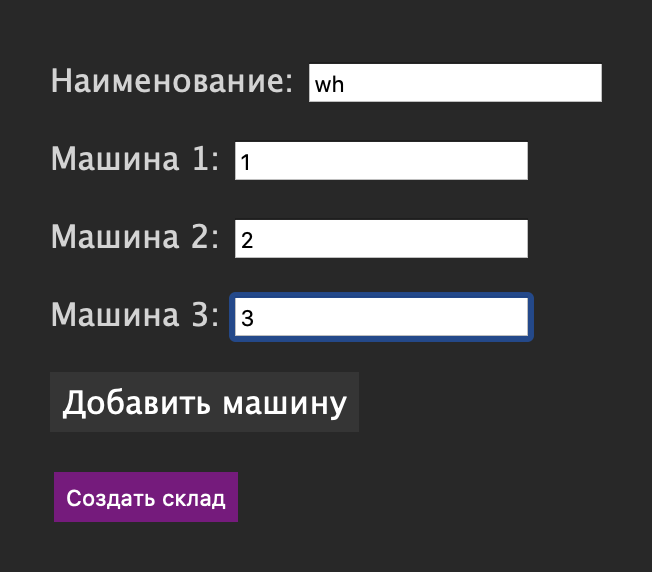
****

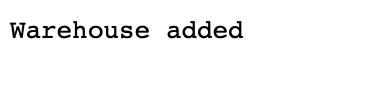
****

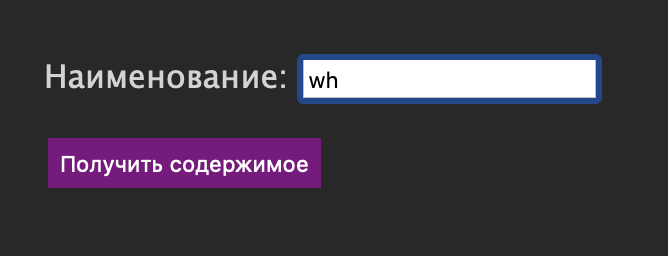
****

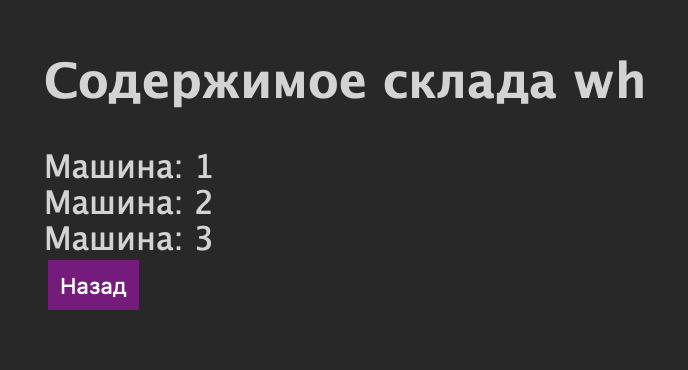
****

****

****

****

****

****

**Задание 2**

Написать скрипт, который принимает на вход число и считает его факториал. Скрипт должен получать параметр через process.argv.

Написать скрипт, который принимает на вход массив чисел и выводит на экран факториал каждого числа из массива. Скрипт принимает параметры через process.argv.

При решении задачи вызывать скрипт вычисления факториала через execSync.

**Реализация**

**fact.js**

"use strict";

const value = process.argv[2];

const num = parseInt(value)

let res = 1;

for (let i = 2; i <= num; i++) {

res \*= i;

}

console.log(res);

**index.js**

"use strict";

const execSync = require('child\_process').execSync;

const values = process.argv.slice(2)

function useCmd(s) {

const options = {encoding: 'utf8'};

const cmd = s.toString();

const answer = execSync(cmd, options);

return answer.toString();

}

function getFact(num) {

const command = `node fact.js ${num}`;

return parseInt(useCmd(command));

}

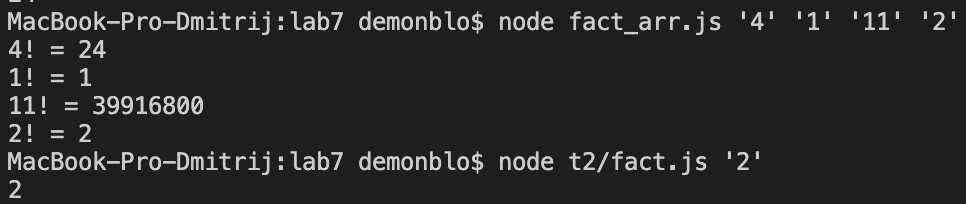
for (const value of values) {

let fact = getFact(value);

console.log(`${value}! = ${fact}`);

}

**Проверка работоспособности**

****

**Вывод**

Язык программирования JavaScript позволяет осуществлять взаимодействие между различными веб-серверами. Возможно использовать скрипты внутри других скриптов при помощи дочерних процессов.

Язык программирования Prolog позволяет описывать алгоритмы при помощи логических операций, что может быть полезно при решении нестандартных задач и лингвистическом анализе.