Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации «Сибирский Государственный Университет Телекоммуникаций и Информатики» СибГУТИ

Курсовая работа по дисциплине

«Операционные системы»

Тема «Диспетчер Задач»

с использованием сокетов

Выполнил: студент 3-го курса, группы ИП-014

Обухов Артем Игоревич

Преподаватель: Нужнов Александр Васильевич

Новосибирск, 2022 г.

**Содержание**

**Введение…………………………………………………………………………………………3**

**Постановка задачи……………………………………………………………………………..4**

**Результат работы программы………………………………………………………………..8**

**Вывод…………………………………………………………………………………………...10**

**Листинг кода…………………………………………………………………………………..11**

**Введение**

Многие приложения, работающие в сети интернет, обмениваются данными по средствам протокола TCP/IP. При работе с протоколом TCP/IP используют интерфейс сокетов.

Диспетчер задач — системный монитор и менеджер запуска. Он предоставляет информацию о производительности компьютера и запущенных приложениях, процессах и использовании ЦП

Сокеты – это интерфейс прикладного программирования для сетевых приложений TCP/IP. Сокеты бывают двух видов: клиентские и серверные(слушающие). Клиентские сокеты устанавливают связь с сервером и обмениваются с ним данными. Серверный сокет обменивается данными с клиентом по уже установленному соединению.

**Постановка задачи**

Разработать сетевое приложение, реализующее диспетчер задач реального времени.

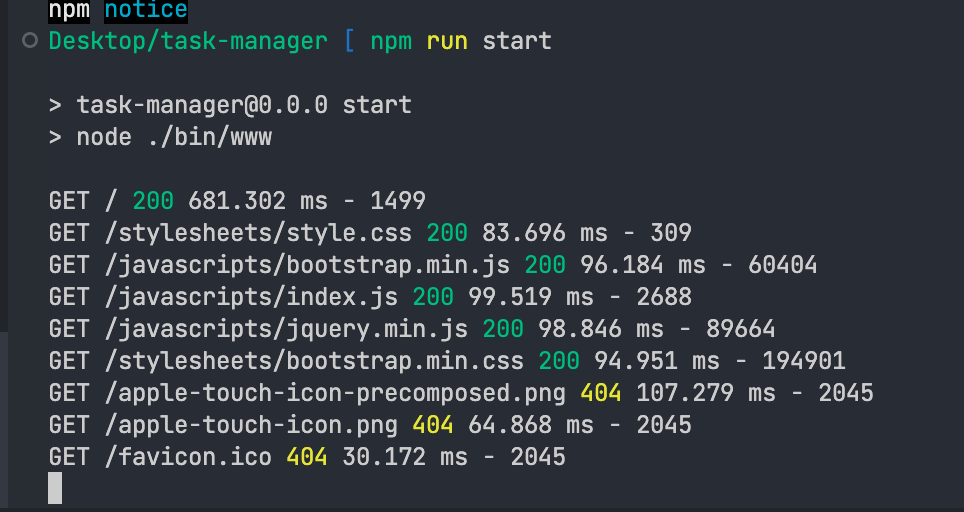
В данной курсовой будет реализован сетевой чат на языке программирования JavaScript и использованием NodeJS и фреймворка Express. Дизайн интерфейса приложения сделан с использованием фреймворка Bootstrap.

Для создания нашего приложения необходимо реализовать два объекта взаимодействия, а именно:

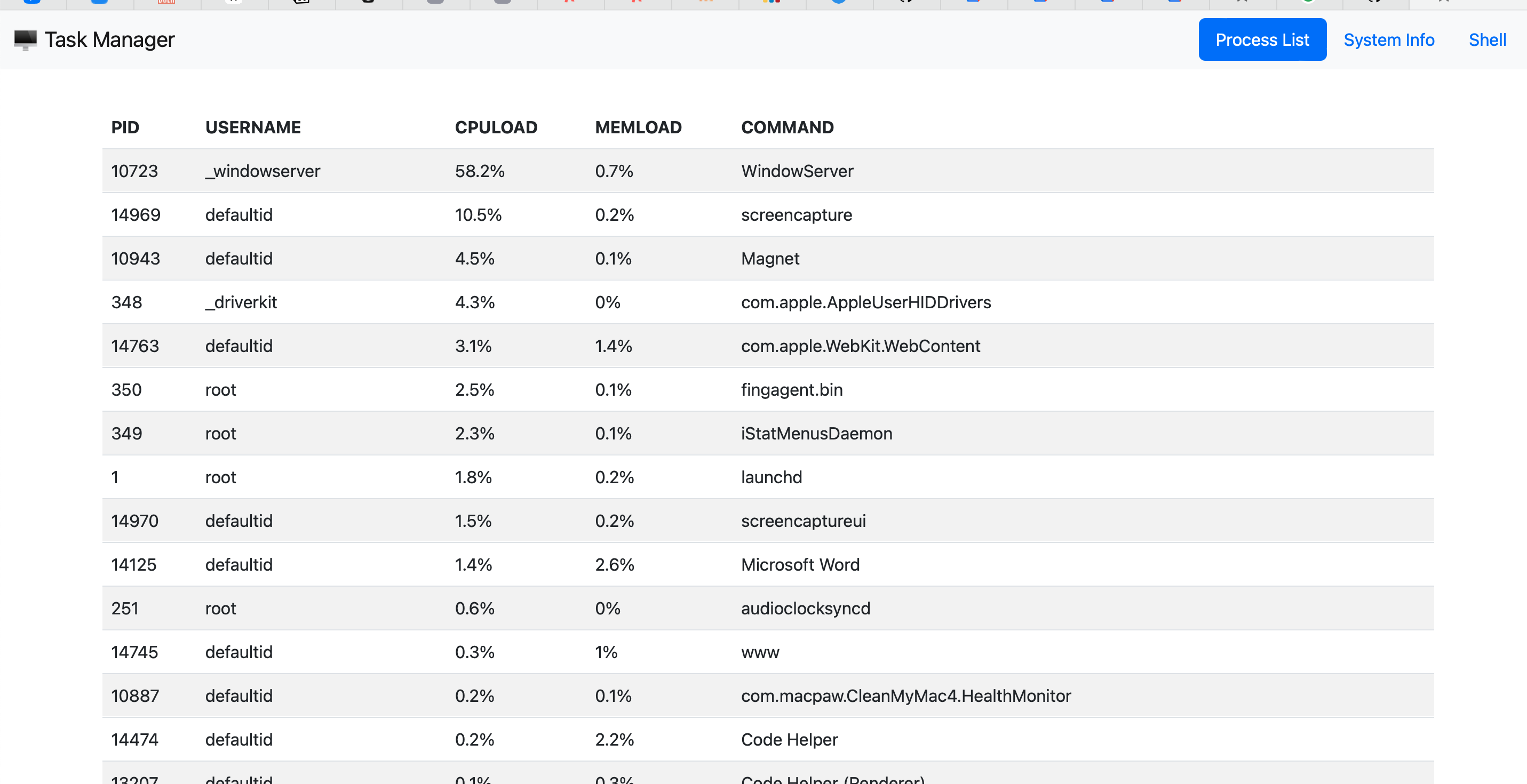
1. Сервер, который буде отправлять сведения о системе в реальном времени.
2. Интерфейс для сохранения подключения сессии Web Shell’a.
3. Клиентскую часть приложения, которая будет иметь постоянно подключение с сервером

**Результаты работы программы**

Запуск сервера:



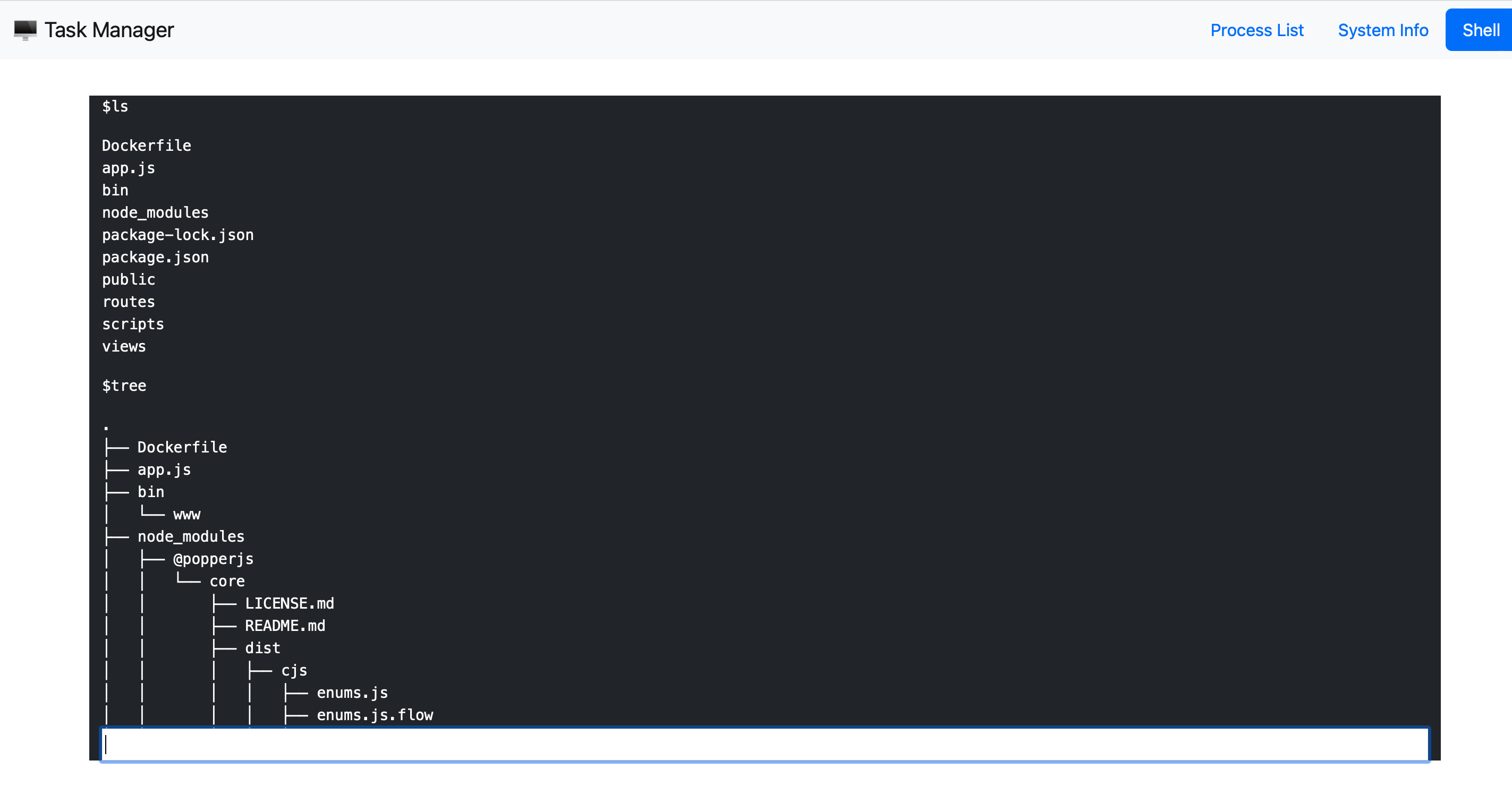
Открытие клиента - на первой вкладке находится список процессов:



Вторая вкладка – общая информация о системе:



Третья вкладка – интерактивный Web Shell



**Вывод**

В ходе работы данной курсовой был реализован диспетчер задач с просмотром списка процессов, отображением общей информации о системе и интерактивным Web Shell’ом

**Листинг кода:**

Сервер:

var createError = require('http-errors'),

express = require('express'),

path = require('path'),

cookieParser = require('cookie-parser'),

logger = require('morgan'),

systemInfo = require('./scripts/systeminfo'),

indexRouter = require('./routes/index'),

app = express(),

http = require('http'),

WebSocketServer = require('ws').Server,

crypto = require('crypto'),

childProcess = require('child\_process');

server = http.createServer(app);

server.listen(8999);

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

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

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

var systemInfoSocket = new WebSocketServer({server: server, path: '/echo'});

systemInfoSocket.on('connection', (ws) => {

var sendSystemInfo = async () => {

let result = await systemInfo();;

ws.send(JSON.stringify(result));

}

setInterval(sendSystemInfo, 1000);

})

var shellSessions = new Map();

server = http.createServer(app);

server.listen(9000);

var shellServer = new WebSocketServer({server: server, path: '/shell'});

shellServer.on('connection', (ws) => {

let shellSession = childProcess.spawn('bash');

ws.on('message', message => {

shellSession.stdin.write(message + "\n");

});

shellSession.stdout.on('data', data => {

ws.send(data.toString());

});

shellSession.stdout.on('error', data => {

ws.send(data.toString());

});

shellSession.stderr.on('data', data => {

ws.send(data.toString());

})

shellSessions.set(ws, shellSession);

});

shellServer.on('close', (ws) => {

console.log('Closed connection')

shellSessions.remove(ws);

});

shellServer.on('error', (ws) => {

console.log('Closed connection')

shellSessions.remove(ws);

});

app.use('/', indexRouter);

// catch 404 and forward to error handler

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

next(createError(404));

});

// error handler

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

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

module.exports = app;

// systeminfo.js

const os = require('os');

const { promisify } = require('util');

const exec = promisify(require('child\_process').exec)

const si = require('systeminformation');

async function getProcessesList() {

let processList = await (await si.processes()).list;

let processListMapped = [];

for (let process of processList) {

mappedProcess = {}

mappedProcess.PID = process.pid;

mappedProcess.userName = process.user;

mappedProcess.cpuLoad = process.cpu + '%';

mappedProcess.memoryLoad = process.mem + '%';

mappedProcess.command = process.command;

processListMapped.push(mappedProcess);

}

return processListMapped;

}

async function getSystemInfo() {

var systemData = {}

systemData.freeMemory = os.freemem();

systemData.hostName = os.hostname();

systemData.cpus = os.cpus().map((object) => {

return {

model: object.model,

speed: object.speed,

}

});

systemData.averageLoad = os.loadavg();

systemData.processesList = await getProcessesList();

return systemData;

}

module.exports = getSystemInfo;

// index.js

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

res.render('index', { title: 'Task Manager' });

});

module.exports = router;

Клиент:

const socketProtocol = (window.location.protocol === 'https:' ? 'wss:' : 'ws:')

const echoSocketUrl = socketProtocol + '//' + window.location.hostname + ":8999" + '/echo'

const socket = new WebSocket(echoSocketUrl);

socket.onopen = () => {

socket.send('Connection established!');

};

socket.onmessage = message => {

let fetchedData = JSON.parse(message.data);

let processList = fetchedData.processesList;

$('tbody').empty();

for (let process of processList) {

let row = '<tr>';

row += '<td>' + process.PID + '</td>';

row += '<td>' + process.userName + '</td>';

row += '<td>' + process.cpuLoad + '</td>';

row += '<td>' + process.memoryLoad + '</td>';

row += '<td>' + process.command + '</td>';

$('tbody').append(row);

}

$('.sys').empty();

let generateRow = (fieldName, value) => {

let row = '<div class="row alert alert-dark">';

row += '<div class="col">' + fieldName + '</div>';

row += '<div class="col">' + value + '</div>';

row += '</div>';

return row;

}

const freeMemory = fetchedData.freeMemory;

const hostname = fetchedData.hostName;

$('.sys').append(generateRow('HOSTNAME', hostname));

let i = 0;

for (let cpu of fetchedData.cpus) {

$('.sys').append(generateRow('CPU MODEL' + (i), cpu.model));

$('.sys').append(generateRow('CPU SPEED' + (i), cpu.speed));

i++;

}

$('.sys').append(generateRow('FREE MEMORY', freeMemory));

}

$('#system-info').hide();

$('#shell').hide();

$('.nav-link').on('click', function(e) {

$('.tab').hide();

$('.nav-link').removeClass('active');

$(this).addClass('active');

let tabId = $(this).data('tab');

$('#' + tabId).show();

});

const shellSocketUrl = socketProtocol + '//' + window.location.hostname + ":9000" + '/shell';

const shellSocket = new WebSocket(shellSocketUrl);

shellSocket.onopen = () => console.log('Connection established!');

shellSocket.onmessage = (message) => {

let appendedData = '<pre>' + message.data + '</pre>';

$('#shell-out').append(appendedData);

let scroll\_to\_bottom = document.getElementById('shell-out');

scroll\_to\_bottom.scrollTop = scroll\_to\_bottom.scrollHeight;

console.log(message.data);

};

$('#console-input').keydown(function (event) {

let keyPressed = event.keyCode || event.which;

if (keyPressed === 13) {

let data = $(this).val();

if (data.trim() == 'clear') {

$('#shell-out').empty();

} else {

shellSocket.send($(this).val());

}

$('#shell-out').append('<pre>$' + data + '</pre>');

$(this).val('');

}

});