Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6**

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ А.К. Погребников

подпись, дата инициалы, фамилия

Студент КИ15–16Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

# Задача лабораторной работы

Используя материалы «Full-Stack Web Development» создать клиентское приложение чата с возможностью мониторинга комнат.

# Листинг программы

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Чат</title>

<meta name="author" content="https://nikita.tk">

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

<link type="text/css" rel="stylesheet" href="index.css"/>

<script type="text/javascript" src="index.js"></script>

</head>

<body>

<h1 class="header">Чат</h1>

<div class="messages"></div>

<input class="input" type="text"/>

</body>

</html>

index.css

body {

}

.header {

text-align: center;

width: 50%;

font-family: cursive;

border-style: double;

}

.messages {

width: 50%;

border-style: double;

}

.message {

border-bottom-style: dotted

}

.message .avatar {

float: left;

width: 60px;

height: 60px;

background-size: 100%;

background-repeat: no-repeat;

}

.message .body {

overflow: hidden;

}

.message .text {

display: inline-block;

padding: 9px;

font-family: cursive;

}

.message .text:before, .message .text:after {

content: " ";

}

.message .headline {

font-size: 10pt;

}

.message .name {

font-weight: bold;

}

.input {

border-style: double;

width: 50%;

background: #eee;

height: 24px;

font: inherit;

}

index.js

const host = `ws://${ location.hostname }:57772`;

const myName = "Тимофей";

const myAvatar = "https://pp.userapi.com/c840227/v840227058/862e8/ZDaOiXuBcL8.jpg";

let URL = host + "/csp/chat/CHAT.WebSocket.cls",

ws = new WebSocket(URL);

ws.addEventListener("open", () => ws.send(JSON.stringify({

name: myName,

avatar: myAvatar

})));

ws.addEventListener("error", (err) => printMessage({

name: "System",

text: "Ошибка соединения: " + err.toString()

}));

ws.addEventListener("close", () => printMessage({

name: "System",

text: "Соединение с сервером завершено"

}));

ws.addEventListener("message", (m) => {

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

if (message["error"])

return console.error(`Сервер записал ошибку: ${ message.error }`);

if (message["updates"] instanceof Array) message["updates"].forEach(update => {

if (update.type === "message")

printMessage(update);

else if (update.type === "notification")

printMessage(update);

else

console.warn("Необработанное сообщение сокета", message);

});

});

function printMessage ({ date = Date.now(), name, text, avatar = "" }) {

let block = document.querySelector(".messages");

block.innerHTML += `<div class="message">

<div class="avatar" style="background-image: url(${ avatar })"></div>

<div class="body">

<div class="headline">

<span class="date">${ new Date(date).toLocaleString() }</span>,

<span class="name">${ name }</span>

</div>

<div class="text">${ text }</div>

</div>

</div>`;

document.body.scrollTop = 99999999;

}

document.addEventListener("DOMContentLoaded", () => {

const input = document.querySelector(".input");

input.addEventListener("keydown", (event) => {

if (input.value && event.keyCode === 13) {

ws.send(JSON.stringify({ "text": input.value }));

input.value = "";

}

});

});

CHAT.WebSocket.cls

Class CHAT.WebSocket Extends %CSP.WebSocket  
{  
  
Parameter MaxLastMessages = 10;  
  
Parameter DefaultAvatar = "svg/person.svg";  
  
Parameter SystemAvatar = "svg/display.svg";  
  
Parameter PushUpdatesInterval = 0.5;  
  
Property LastSentMessageId [ InitialExpression = 0 ];  
  
Property MyName [ InitialExpression = "AnonymousUser" ];  
  
Property MyAvatar [ InitialExpression = {..#DefaultAvatar} ];  
  
Method OnPreServer() As %Status  
{  
    if ($get(^ChatWebSocket("messages")) = "") {  
     set ^ChatWebSocket("messages") = 0  
    }  
    return $$$OK  
}  
  
Method Server() As %Status  
{  
  
    while (1) { // Бесконечный цикл до тех пор пока клиент не отключится  
  
     set data = ..Read(, .status, ..#PushUpdatesInterval)  
     // Если статус не является ошибкой таймаута (например, клиент отключился)  
     if ($$$ISERR(status)) && ($$$GETERRORCODE(status) '= $$$CSPWebSocketTimeout) {  
     quit  
     }  
     if (data '= "") { // Если поступили какие-либо данные от клиента  
     set err = ..ProcessClientData(data)  
     do:(err '= "") ..ReportError(err)  
     }  
     // Проверика на то, появилось ли новое сообщение за интервал прослушивания (PushUpdatesInterval)  
     continue:(^ChatWebSocket("messages") = ..LastSentMessageId) // Возврат в начало цикла, если не появилось  
     // Когда поступило новое соообщение - перебрать все сообщения и отправить клиенту в количестве MaxLastMessages  
     set lastId = $order(^ChatWebSocket("messages", ""), -1)  
     set limit = ..#MaxLastMessages  
     set messages = []  
     set id = lastId  
     while (id > ..LastSentMessageId) && (limit > 0) {  
     set message = ^ChatWebSocket("messages", id)  
     set avatar = $LISTGET(message, 4)  
     do messages.%Push({  
     "type": "message",  
     "date": ($LISTGET(message, 1)),  
     "name": ($LISTGET(message, 2)),  
     "text": ($LISTGET(message, 3)),  
     "avatar": ($case(avatar = "", 1: ..#DefaultAvatar, :avatar))  
     })  
     set limit = limit - 1  
     set id = $order(^ChatWebSocket("messages", id), -1)  
     }  
     set ..LastSentMessageId = lastId  
     // Инверсия и сортировка по дате и времени сообщений  
     set reversedMessages = []  
     for i=messages.%Size()-1:-1:0 {  
     do reversedMessages.%Push(messages.%Get(i))  
     }  
     // Отправить сформированные данные на клиент  
     do ..Send({  
 "updates": (reversedMessages)  
 })  
    }  
    return $$$OK  
}  
  
Method ProcessClientData(data) As %String  
{  
 set json = ""  
 // Расшифровка сообщения от клиента  
 try {  
 set json = ##class(%DynamicObject).%FromJSON(data)  
 } catch (e) {  
 set err = "Unable to parse incoming message: is it a valid JSON format?"  
 set ^ChatWebSocket("errors", $I(^ChatWebSocket("errors"))) = $LB(err, e)  
 return err  
 }  
 return:(json = "") "Невозможно расшифровать входящее сообщение от клиента: это JSON формат?"  
 // Обновление данных клиента и отправка нового сообщения  
 set nameIsSet = (json.name '= "") && ('$IsObject(json.name))  
 set avatarIsSet = (json.avatar '= "") && ('$IsObject(json.avatar))  
 set textIsSet = (json.text '= "") && ('$IsObject(json.text))  
 set:nameIsSet ..MyName = $EXTRACT(json.name, 1, 30)  
 set:avatarIsSet ..MyAvatar = json.avatar  
 // Если задан аватар и имя, но нет сообщения (в случае первого входа)  
 if ((avatarIsSet || nameIsSet) && ('textIsSet)) {  
 do ..Send({  
 "updates": [{  
 "type": "notification",  
 "name": "System",  
 "avatar": (..#SystemAvatar),  
 "text": "Добро пожаловать в чат!"  
 }]  
 })  
 return ""  
 }  
 return:('textIsSet) "Невозможно отправить сообщение:"  
 \_ " введите что-либо в текстовое поле"  
 set ^ChatWebSocket("messages", $I(^ChatWebSocket("messages"))) = $LB(  
 $ZDATETIME($HOROLOG), // Текущее время  
 ..MyName,  
 json.text,  
 $case(..MyAvatar = ..#DefaultAvatar, 1: "", :..MyAvatar)  
 )  
  
 return ""  
}  
  
Method ReportError(err = "")  
{  
 return:(err = "")  
 do ..Send({  
 "error": (err),  
 "updates": []  
 })  
}  
  
Method Send(jsonObj)  
{  
 return ..Write((jsonObj).%ToJSON())  
}  
  
}

# Описание программы

На рисунке 1 можно ознакомиться с реализованной комнатой чата.

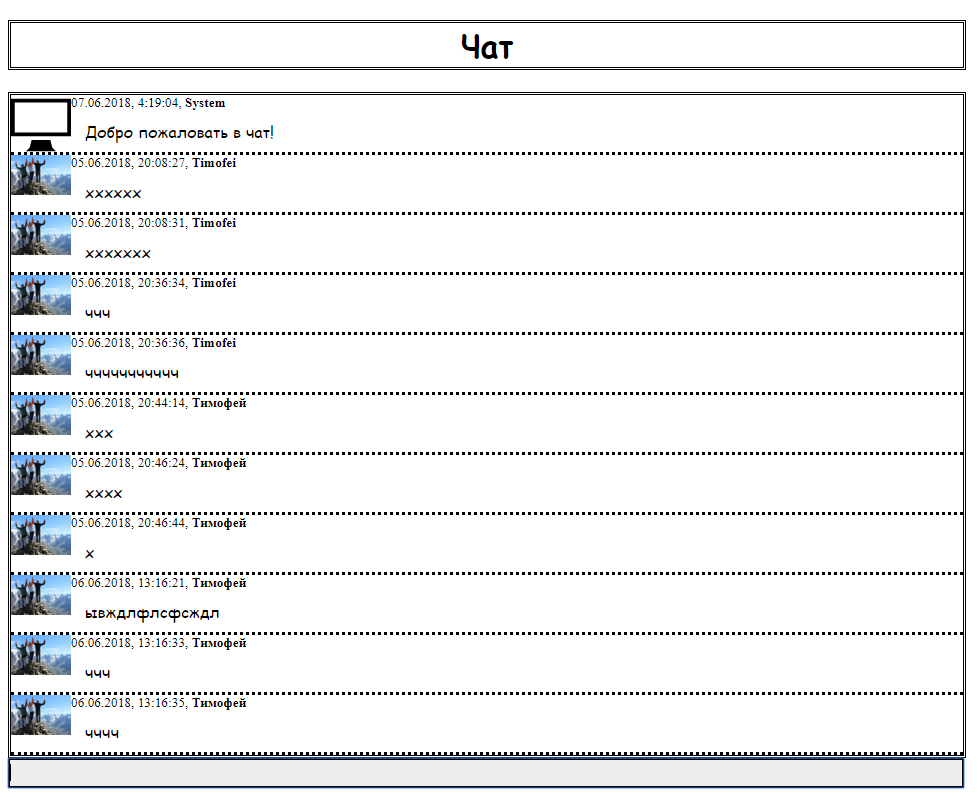


Рисунок 1 – Структура

Создается соединение с сокетом через URL, состоящем из адреса хоста и пути к классу-сокету. Сокет на стороне сервера получаем через название (которое определяет комнату чата, в данном случае это сокет с названием «messages»). Сокет хранит в себе сообщения: клиент может как добавлять туда новые сообщения, так и получать в том количестве и с тем интервалом, которые заданы на сервере. Обмен совершается через JSON-строки.

Мониторинг комнат осуществляется через извлечение всех глобалов ^ChatWebSocket (комнат) из базы данных и подключением к нужному по названию (например, имени пользователя) и отправке туда сообщений. Так как пользователь на локальной машине один, то и комната создается одна.