УДК 004.4

Разработка веб-чата в закрытой локальной сети на основе языка программирования Node.js

Якимов Антон Сергеевич
Приамурский государственный университет им. Шолом-Алейхема
Магистрант

Пасюков Александр Андреевич
Приамурский государственный университет им. Шолом-Алейхема
Магистрант

Николаев Сергей Валерьевич
Приамурский государственный университет им. Шолом-Алейхема
Магистрант

Баженов Руслан Иванович
Приамурский государственный университет им. Шолом-Алейхема
к. п. н., доцент, зав. кафедрой информационных систем, математики и методик обучения

Аннотация
Во многих средних и крупных компаниях закрывают доступ в интернет в связи с безопасностью компьютерных сетей. Этим усложнилось общение между сотрудников компании удаленных отделов. Таким образом, в данной статье описывается разработка веб-чата для обмена сообщениями в реальном времени между сотрудниками из разных удаленных отделов в закрытой сети предприятия.
Ключевые слова: локальная сеть, web-чат, Node.js, JavaScript.

Development of web-chat in a closed local network based on the Node.js programming language

Yakimov Anton Sergeevich
Sholom-Aleichem Priamursky State University
Master student

Pasyukov Alexandr Andreevich
Sholom-Aleichem Priamursky State University
Master student

Nikolaev Sergey Valerievich
Sholom-Aleichem Priamursky State University
Master student

Bazhenov Ruslan Ivanovich
Sholom-Aleichem Priamursky State University
Candidate of pedagogical sciences, associate professor, Head of the Department of Information Systems, Mathematics and teaching methods

Abstract
In many medium and large companies, due to the security of computer networks, they block Internet access. This complicated the communication between employees of the company's remote departments. Thus, this article describes the development of a live chat web chat for real-time messaging between people from different remote departments in a closed enterprise network.
Keywords: local network, web-chat, Node.js, JavaScript.

Вопросы в плане общения всегда интересовали человечество. Для обмена информацией на огромных расстояниях люди изобрели почтовые системы, интернет и локальные линии связи. При развитии связи были изобретены чаты, которые обеспечили передачу текстовых сообщений в любую точку мира, имея требуемое программное обеспечение. К примеру, во многих средних и крупных компаниях в связи с организацией безопасности важных компьютерных сетей путем закрытия доступ в интернет затрудняется связь между сотрудников в удаленных отделах. Таким образом, было решено разработать веб-чат для обмена сообщениями в реальном времени между людьми из разных удаленных отделов в закрытой сети предприятия.

Многие русские и зарубежные ученые занимались данной проблемой. Л. М. Козополянская и А. И. Газейкина [1] в своей работе описали разработку клиент-серверного приложения с использованием сокетов на языке JAVA. В статье П. А. Васильев [2] кратко описана библиотека Socket.io языка программирования Node.js. В работе З. С. Сейдаметова и др.[3] описаны разработанные системы мгновенного обмена сообщения, а также описано проектирование и разработка приложения ATALK. D.Hausen и S. Boring [4] описали разработку системы мгновенной передачи сообщений в городском управлении. A. Nalawade и D. Kamdar [5] раскрыли плюсы интегрированной системы обмена сообщений. L.Xiao и V. Mazalov [6] описали плюсы мессенджеров при обмене сообщений.

При разработке программного продукта были выявлены следующие требования:

1. Должен работать в сети без выхода в интернет.
2. Должен иметь возможность авторизации по логину и паролю.
3. Должен уметь обмениваться сообщениями в реальном времени.
4. Должен отображать список сотрудников, находящихся в сети.
5. Должен уметь отправлять сообщения как одному сотруднику, так и целому отделу.

Для решения данной задачи за основу был взят язык программирования Node.js. Node,js - программная платформа, основанная на движке V8

разработанном на основе языка JavaScript. Данный язык превращает JavaScript из узкоспециализированного языка в язык общего назначения. За счет подключения дополнительных библиотек, написанных на разных языках, получает большую универсальность.

После установки Node.js была установлена библиотека Socket.IO. Данная библиотека предназначена для передачи данных между web-приложениями (клиентской и серверной) в реальном времени. Также были установлены и подключены дополнительные модули, такие как express, log4js, http. На рисунке 1 изображено подключение дополнительных модулей.

```
var express = require('express'); // Подключаем express
var app = express();
var server = require('http').Server(app); // Подключаем http через app
var io = require('socket.io')(server); // Подключаем socket.io и указываем на сервер
var log4js = require('log4js'); // Подключаем наш логгер
var logger = log4js.getLogger(); // Подключаем с модуля log4js сам логгер
```

Рисунок 1. Подключение дополнительных модулей

Далее, после подключения дополнительных модулей, требуется подключиться к серверу. Для этого создаем переменную, куда помещаем порт для прослушивания, и запускаем логирование и прослушку данного порта. После чего, переходя по адресу «localhost:3000», он будет отображать нашу страницу с нашим чатом. На рисунке 2 изображено подключение к серверу.

```
var port = 3000; // Указываем порт
var socket = io.connect('http://localhost:' + port); // подключаемся к серверу через порт
logger.debug('Script has been started...'); // логгируем
server.listen(port); // теперь подключаемся к серверу через порт 3000
```

Рисунок 2. Подключение к серверу

В данном случае «io.connect» создает событие «connection», далее требуется подключить обработчика на стороне сервера, в противном случае подключение будет прервано, но в нашем случае требуется привязка логина к авторизированным пользователям. На рисунке 3 изображено создание обработчика.

```
io.on('connection', function (socket) { // Создаем обработчик события 'connection'
var name = 'U' + (socket.id).toString().substr(1,4); // Создаем никнейм нашему клиенту.
socket.broadcast.emit('newUser', name); // Отсылает событие 'newUser' всем подключенным, кроме текущего.
socket.emit('userName', name); // Отправляем текущему клиенту событие 'userName' с его ником
logger.info(name + ' connected to chat!'); // Логгирование
});
```

Рисунок 3. Создание обработчика

При открытии адреса «localhost:3000» отображается сообщение приветствия с нашим логином. Далее требуется организовать действие через кнопку «Отправить» с помощью jQuery для отправки сообщения. После чего при клике на данную кнопку, текст будет отправляться на сервер. На рисунке 4 изображена отправка сообщений через кнопку на сервер.

```
$(document).on('click', 'button', function(){ // Прослушка кнопки на клик
var message = $('input').val(); // Все что в поле для ввода записываем в переменную
socket.emit('message', message); // Отправляем событие 'message' на сервер
$('input').val(null);
});
```

Рисунок 4. Отправка сообщений через кнопку на сервер

На следующем шаге требуется сделать обработчик события на самом сервере. Таким образом, после отправки текста со страницы, он будет логироваться в консоли Node.js. На рисунке 5 изображен обработчик события на стороне сервера.

```
io.on('connection', function (socket) {
    var name = 'U' + (socket.id).toString().substr(1,4);
    socket.broadcast.emit('newUser', name);
    socket.on('message', function(msg){
      logger.warn('-----------');
      logger.warn('User: ' + name + ' | Message: ' + msg);
      logger.warn('====> Sending message to other chaters...');
      io.sockets.emit('messageToClients', msg, name); // Отправляем текст и логин отправителя
    });
  });
```

Рисунок 5. Обработчик события на стороне сервера

Последним шагом для полноценной работы чата будет организация обработчика события со стороны клиента. После этого клиент сможет получить возможность принимать сообщения с сервера. На рисунке 6 изображен обработчик события на стороне клиента.

```
socket.on('messageToClients', function(msg, name){
    console.log(name + ' | => ' + msg); // Логгирование в консоль браузера
    $('textarea').val($('textarea').val() + name + ' : '+ msg +'\n');
    });
```

Рисунок 6. Обработчик события на стороне клиента

Немного настроив чат под требования компании, чат получил возможность ведения переписки с несколькими сотрудниками одновременно в разных каналах, и добились полноценного чата для закрытой сети. На рисунке 7 изображен внешний вид чата

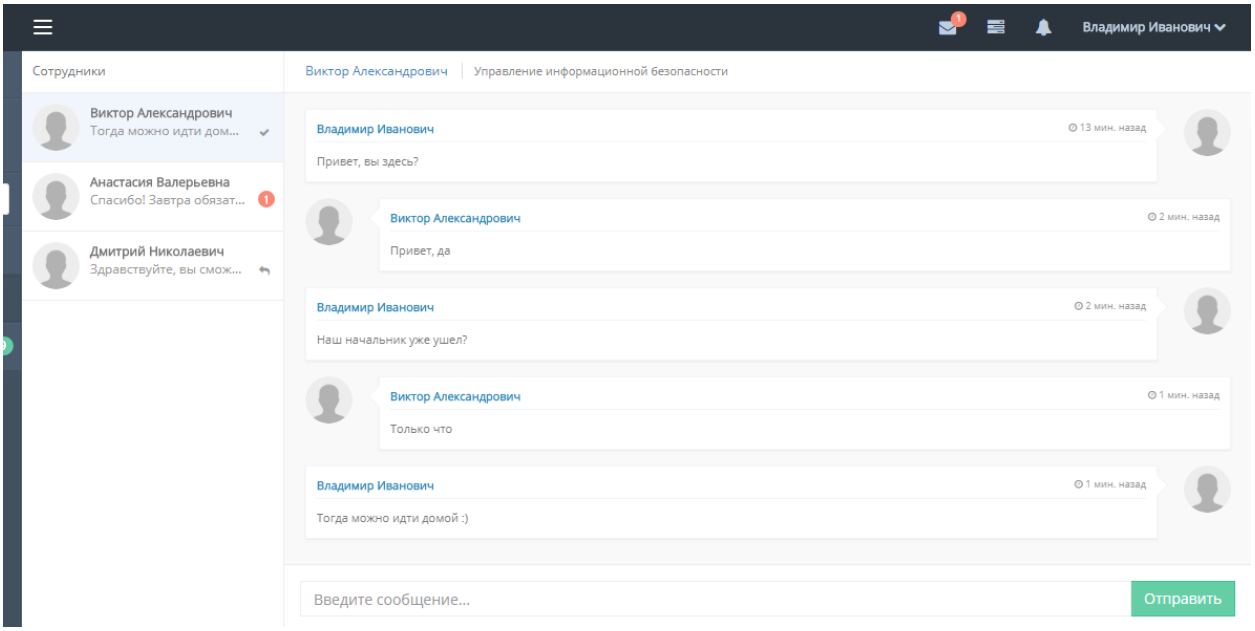

Рисунок 7. Внешний вид чата

Далее для удобства было решено организовать поиск по сотрудникам. На рисунке 8 изображена страница поиска.

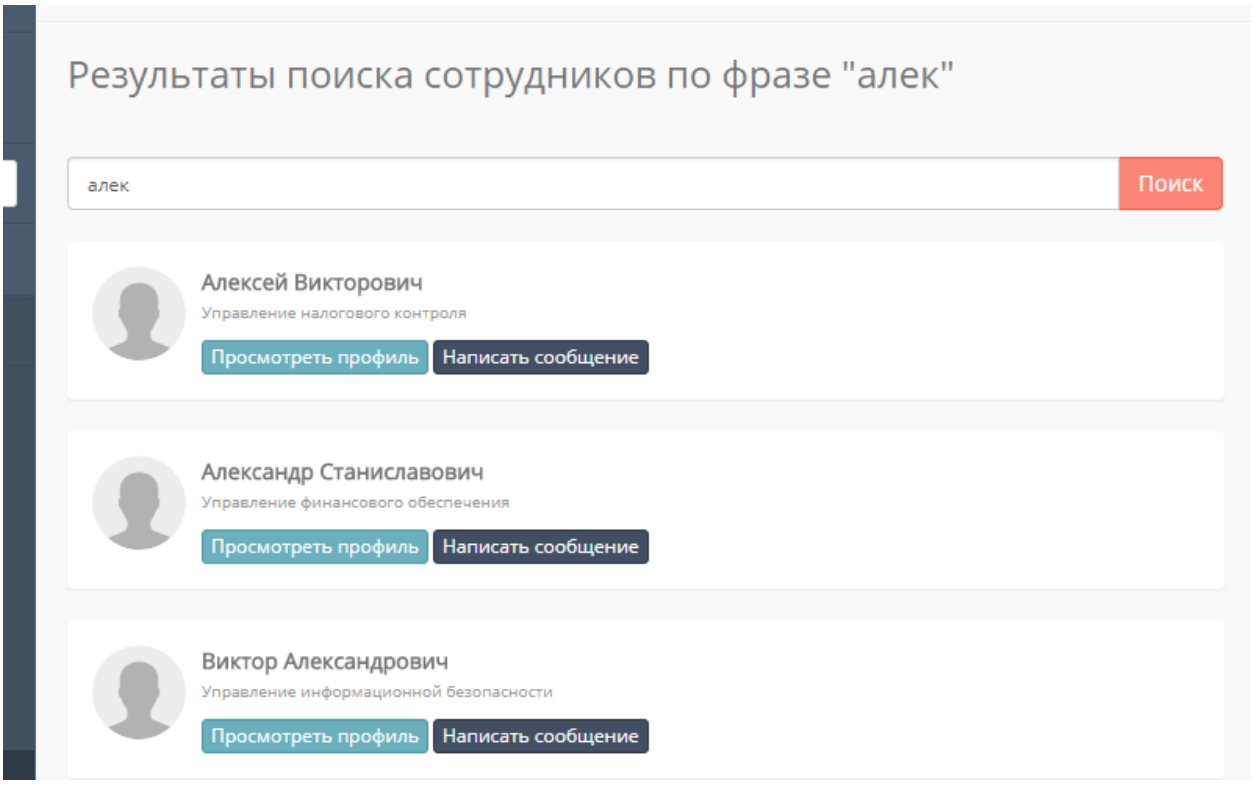

Рисунок 8. Страница поиска

В результате данной работы был разработан web-чат для работы в закрытой локальной сети. В данной статье описана разработка основного функционала программы. Таким образом, данной программой можно пользоваться не только в закрытых локальных сетях, но и через интернет при

определенной настройке. В дальнейшем планируется усовершенствование данной системы для возможности передачи текстовых файлов и т.д.

Библиографический список

1. Козополянская Л. М., Газейкина А. И. Разработка клиент-серверного приложения с использованием сокетов на языке JAVA //Актуальные вопросы преподавания математики, информатики и информационных технологий. 2016. С. 45-49.
2. Васильев П. А. Библиотека Socket. io языка программирования Node JS // Научные исследования. 2016. С. 25-26.
3. Сейдаметова З. С., Асанова У. Б., Костина Е. Г. Системы мгновенного обмена сообщения: проектирование и разработка приложения ATALK //Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2016. №. 2. С. 5-21.
4. Hausen D., Boring S., Lueling C., Rodestock S., Butz A. StaTube: Facilitating state management in instant messaging systems // Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (Kingston, Ontario, Canada, February 19-22, 2012). USA, New York, NY: ACM, 2012. pp. 283-290.
5. Nalawade A., Kamdar D., Angolkar P., Gaikwad S. Integrated instant messaging system // International Journal of Latest Trends in Engineering and Technology (IJLTET). 2014. Vol. 3, issue 3. pp. 351-355.
6. Xiao L., Mazalov V. Message visualizer: a visualization tool for chat messages // Proceedings of the 2012 iConference (Toronto, ON, Canada, February 7-10, 2012). USA, New York, NY: ACM, 2012. pp. 426-428.