МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра прикладного и системного программирования

КУРСОВАЯ РАБОТА

по дисциплине «Технологии взаимодействия с базами данных»

**разработка веб-приложения «Online chat»**

Величко Дмитрий Марьянович,

3 курс, группа 34з

Руководитель:

Сергеенко Сергей Владимирович,

старший преподаватель

Витебск, 2022

РЕФЕРАТ

Отчет по курсовой работе: с. 26, рис 8, ист. 5, прил. 2.

ПРОГРАММНЫЙ ПРОДУКТ, САЙТ, СРУКТУРА, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ПРОЕКТ, ИНТЕРФЕЙС, СРЕДСТВА РАЗРАБОТКИ, ПРОЕКТИРОВАНИЕ, ТЕСТИРОВАНИЕ.

Целью работы является разработка веб-приложения «Online Chat».

Объектом и предметом исследования являются технологии: ReactJS, ReduxJS, NodeJS, ExpressJS.

В первом разделе содержится описание предметной области и описание используемых технологий.

Во втором разделе описывается практическая реализация проекта.

Автор проекта подтверждает, что приведённый в нём расчётно-аналитический материал правильно и объективно отражает состояние исследуемого процесса, а все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их автора

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc62468939)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc62468940)

[1.1 Язык программирования JavaScript 5](#_Toc62468941)

[1.2 Технология ReactJS 6](#_Toc62468942)

[1.3 Технология ReduxJS 6](#_Toc62468943)

[1.4 Технология NodeJS 7](#_Toc62468944)

[1.5 Технология ExpressJS 8](#_Toc62468944)

[2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОЕКТА 9](#_Toc62468945)

[2.1 Формирование структуры проекта 9](#_Toc62468946)

[2.2 Алгоритмическое обеспечение проекта 9](#_Toc62468947)

[2.3 Реализация программного продукта 10](#_Toc62468948)

[ЗАКЛЮЧЕНИЕ 1](#_Toc62468949)6

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 1](#_Toc62468950)7

[ПРИЛОЖЕНИЕ А 1](#_Toc62468951)8

[ПРИЛОЖЕНИЕ Б 2](#_Toc62468952)6

# ВВЕДЕНИЕ

Уже несколько десятков лет компьютер занял большую часть нашей повседневной жизни, и многие не представляют своей учёбы, работы без компьютера. В настоящее время трудно назвать те области человеческой деятельности, успехи в которых не были бы связаны с использованием компьютера. Сфера применения компьютера постоянно расширяется, существенно влияя на развитие производительных сил нашего общества. Непрерывно изменяются технико-экономические характеристики компьютера, например, такие, как быстрота действия, ёмкость памяти, надёжность в работе, стоимость, удобства в эксплуатации, габаритные размеры, потребляемая мощность и др. В широком понимании всякий компьютер рассматривается как преобразователь информации.

Целью данного курсового проекта является создание онлайн чата для общения между пользователями и хранения соответствующей информации в базе данных, та же реализовать вход и регистрацию нового пользователя на языке JavaScript (NodeJS, ReactJS, ExpressJS, Redux), с использованием системы управления базами данных (СУБД) MySQL.

При разработке приложения, необходимо решить следующие задачи:

* регистрация пользователя;
* вход в систему;
* отправка сообщения через протокол Websocket;
* вывод информации на экран: текстовой, графической и др.;
* возможность создавать и управлять новыми чатами.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## Язык программирования JavaScript

«JavaScript» – это интерпретируемый язык программирования с объектно-ориентированными возможностями. «JavaScript» – это не типизированный язык, т. е. в нем не требуется определять типы переменных. Объекты в «JavaScript» отображают имена свойств на произвольные значения.

Ядро языка «JavaScript» поддерживает работу с такими простыми типами данных, как числа, строки и булевы значения. Помимо этого, он обладает встроенной поддержкой массивов, дат и объектов регулярных выражений.

Обычно «JavaScript» применяется в веб-браузерах, а расширение его возможностей за счет введения объектов позволяет организовать взаимодействие с пользователем, управлять веб-браузером и изменять содержимое документа, отображаемое в пределах окна веб-браузера. Эта встроенная версия «JavaScript» запускает сценарии, внедренные в «HTML» код веб-страниц. Как правило, эта версия называется клиентским языком «JavaScript», чтобы подчеркнуть, что сценарий исполняется на клиентском компьютере, а не на веб-сервере.

В основе языка «JavaScript» и поддерживаемых им типов данных лежат международные стандарты, благодаря чему обеспечивается прекрасная совместимость между реализациями. Некоторые части клиентского «JavaScript» формально стандартизированы, другие части стали стандартом дефекта, но есть части, которые являются специфическими расширениями конкретной версии браузера. Совместимость реализаций в разных браузерах зачастую приносит немало беспокойств программистам, использующим клиентский язык «JavaScript».

Так же, в проекте использовались ещё многие технологии, основанные на JavaScript, которые описаны далее.

## 1.2 Технология ReactJS

JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

## 1.3 Технология ReduxJS

Redux — библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с React или Angular для разработки клиентской части. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст.

Redux – библиотека с простым API, предсказуемое хранилище состояния приложений. Она работает по тому же принципу, что и функция reduce, один из концептов функционального программирования. Её создатели вдохновлялись функциональным языком программирования Elm.

Эта библиотека реализует шаблон архитектуру Flux.Основной отличительной особенностью Flux является односторонняя направленность передачи данных между компонентами Flux-архитектуры. Архитектура накладывает ограничения на поток данных, в частности, исключая возможность обновления состояния компонентов самими собой. Такой подход делает поток данных предсказуемым и позволяет легче проследить причины возможных ошибок в программном обеспечении.

В минимальном варианте Flux-архитектура может содержать три слоя, взаимодействующие по порядку:

* Actions (действия)
* Stores (хранилища)
* Views (представления)

Хотя обычно между действиями и хранилищами добавляют Dispatcher (диспетчер).

В первую очередь Flux работает с информационной архитектурой, которая затем отражается в архитектуре программного обеспечения, поэтому уровень представлений слабо зацеплен с другими уровнями системы.

Существенные плюсы использования Redux, как системой управления состояния приложения:

* Redux помогает писать приложения, которые ведут себя согласованно, работают в разных средах (клиентских, серверных и собственных) и легко тестируются;
* Централизация состояния и логики вашего приложения обеспечивает такие мощные возможности, как отмена / повтор, сохранение состояния и многое другое;
* Redux DevTools позволяет легко отслеживать, когда, где, почему и как изменилось состояние вашего приложения. Архитектура Redux позволяет вам регистрировать изменения, использовать «отладку во времени» и даже отправлять полные отчеты об ошибках на сервер;
* Redux работает с любым слоем пользовательского интерфейса и имеет большую экосистему надстроек, отвечающих вашим потребностям.

## 1.4 Технология NodeJS

NodeJS — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. NodeJS добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. NodeJS применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на NodeJS и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel, lowJS и espruino). В основе NodeJS лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

C Node проще масштабироваться. При одновременном подключении к серверу тысяч пользователей Node работает асинхронно, то есть ставит приоритеты и распределяет ресурсы грамотнее. Java же, например, выделяет на каждое подключение отдельный поток.

## 1.5 Технология ExpressJS

ExpressJS, или просто Express, фреймворк web-приложений для NodeJS, реализованный как свободное и открытое программное обеспечение под лицензией MIT. Он спроектирован для создания веб-приложений и API. Де-факто является стандартным каркасом для NodeJS. Автор фреймворка, TJ Holowaychuk, описывает его как созданный на основе написанного на языке Ruby каркаса Sinatra, подразумевая, что он минималистичен и включает большое число подключаемых плагинов. Express может являться backend'ом для программного стека MEAN, вместе с базой данных MongoDB и каркасом VueJS, React или AngularJS для frontend'а.

ExpressJS имеет готовые функции обработки HTTP запросов, причем для каждого HTTP метода имеется своя функция, что особенно удобно при создании REST API. И это далеко не единственная причина использования ExpressJS.

# 2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОЕКТА

## 2.1 Формирование структуры проекта

Разработка алгоритма функционирования программного продукта проводилась нисходящим методом, который заключается в том, что разрабатываемый алгоритм разделяется на дочерние алгоритмы, а те, в свою очередь, разделяются на элементарные алгоритмы. Структуру проекта можно представить в виде нескольких блоков. В первом блоке выполняется запуск программы, в ходе которого происходит подключение базы данных к оболочке программы. В втором блоке осуществляется просмотр и редактирование данных (добавление, изменение, удаление). В третьем блоке осуществляется вывод данных на экран. Данный программный продукт, состоит из трех составляющих частей: базы данных, спроектированной в MySQL; веб-сайта, разработанного с помощью языка ReactJS, Redux и сервера, разработанного с помощью NodeJS, ExpressJS.

## 2.2 Алгоритмическое обеспечение проекта

В ходе проектирования программного продукта исследована предметная область, собрана необходимая информация, определены цели и задачи проектирования базы данных. Необходимо выполнить концептуальное, логическое и физическое проектирование базы данных. Интерфейс должен быть простым и удобным для пользователя. При этом веб-сайт должен хранить необходимую информацию и выполнять заявленные функции. Для чего необходимо создать программную оболочку (серверную часть) для созданной базы данных.

При организации базы данных создано 3 таблицы, которые содержит всю необходимую информацию о пользователе как, идентификатор пользователя, логин, пароль и другие данные. Таблица для хранения информации по созданным комнатам (таблица «rooms») и таблица для хранения истории сообщений в каждой комнате (таблица «messages»). Для связей в главной таблице «users», существуют поля «users.id» и «rooms.ownerId». Более подробно схему сущностей, можно посмотреть на изображении (рисунок 2.0).

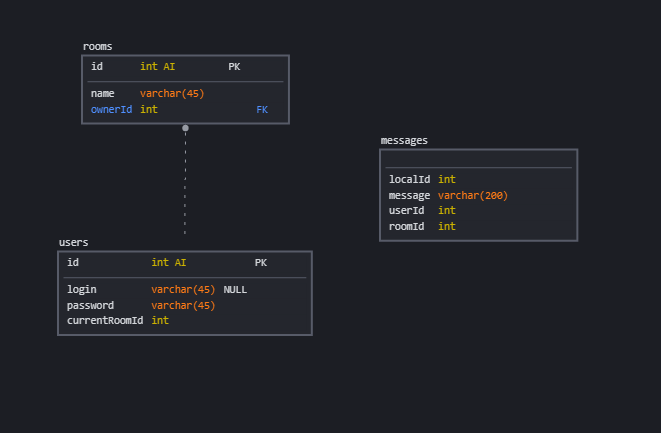


Рисунок 2.0 – Схема сущностей

Путём устранения дублирования данных была спроектированная реляционная база данных, в процессе нормализации, которой обеспечивается защита целостности данных.

## 2.3 Реализация программного продукта

Процесс создания программного продукта делится на несколько стадий:

* изучение выбранной предметной области;
* обработка полученной информации;
* выбор средств реализации;
* программирование;
* тестирование;
* эксплуатация.

Наиболее целесообразный метод тестирования – проверка работоспособности основных функций программы. Программа должна нормально функционировать при внесении информации и т. п. Процесс создания программного продукта делится на несколько стадий: изучение выбранной предметной области; обработка полученной информации; выбор средств реализации; программирование; тестирование; эксплуатация.

Для запуска сайта, необходимо иметь на компьютере или на удалённом сервере NodeJS от 12 версии и выше. В папке проекта есть две директории «server» и «client». Для запуска клиента переходим в соответствующую папку и выполняем в ней команду «npm install» а после «npm start», так же перед этим в «config.ts» нужно поменять URL веб-сервера, для соединения клиента с ним. Что бы запустить сервер нужно так же выполнить две команды «npm install» и «npm start». После запуска сервера, открываем сайт в браузере под ссылкой «localhost:3000». Потом мы видим страницу для входа, изображенная на рисунке 2.1.

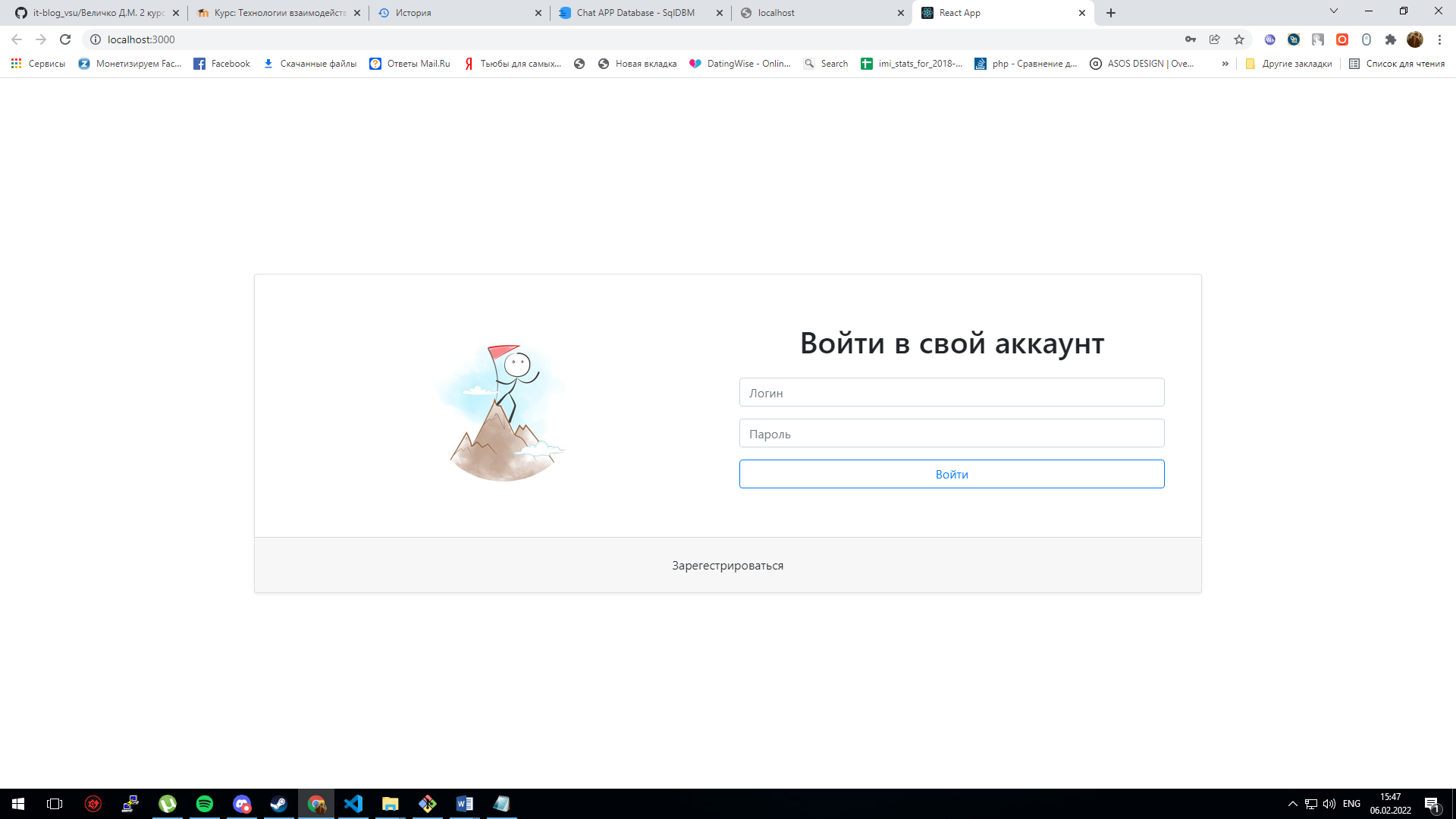


Рисунок 2.1 – Страница для входа

Для начала нам нужно зарегистрироваться, нажав на кнопку «Зарегистрироваться». После мы увидим форму для регистрации. (рисунок 2.2)

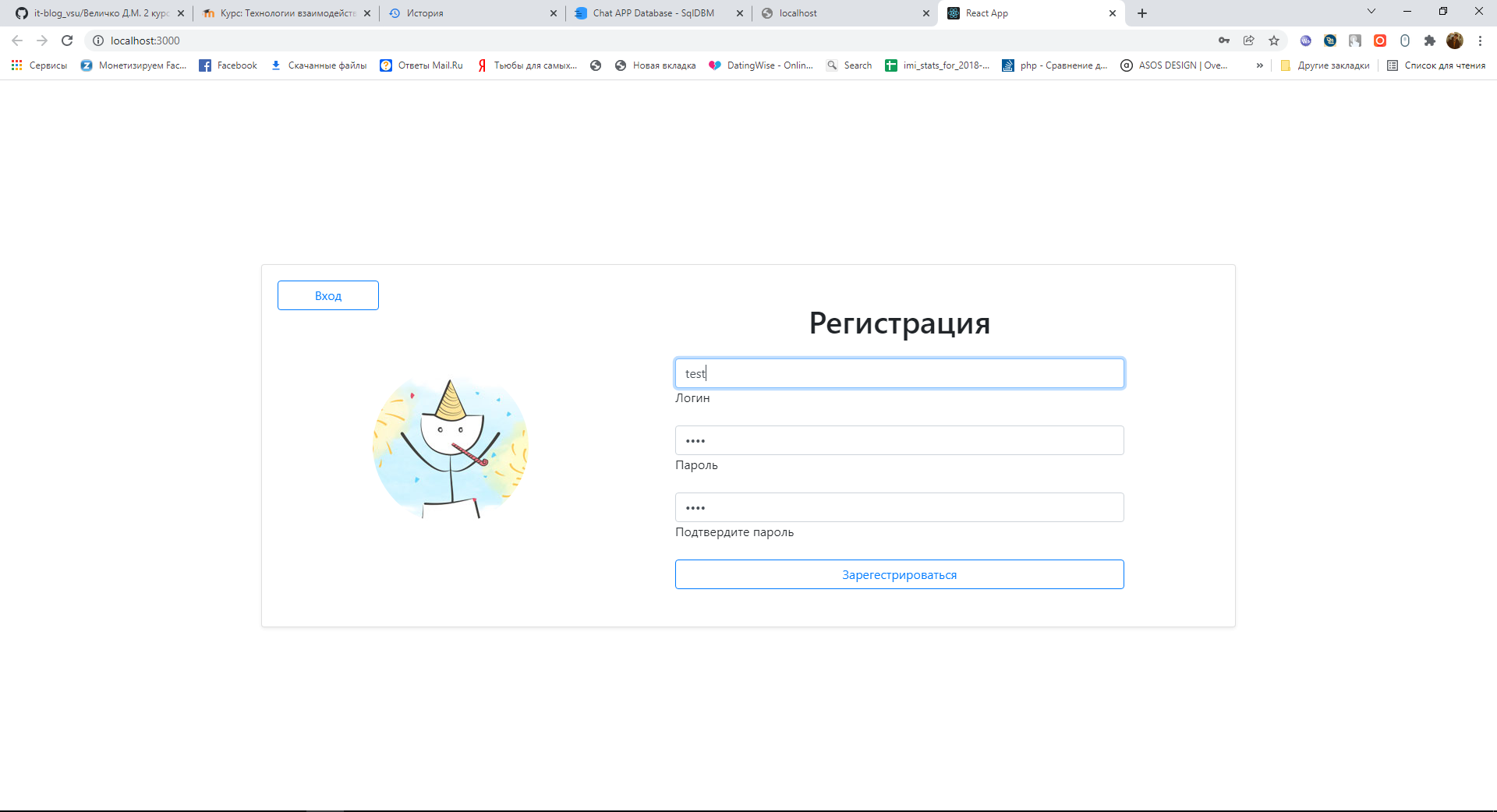


Рисунок 2.2 – Страница регистрации

После регистрации мы можем войти в наш аккаунт. Нажав на кнопку «Войти» в окне с авторизацией. После входа мы видим главную страницу, где отображен перечень текущих чатов (слева) и сам чат (рисунок 2.4)

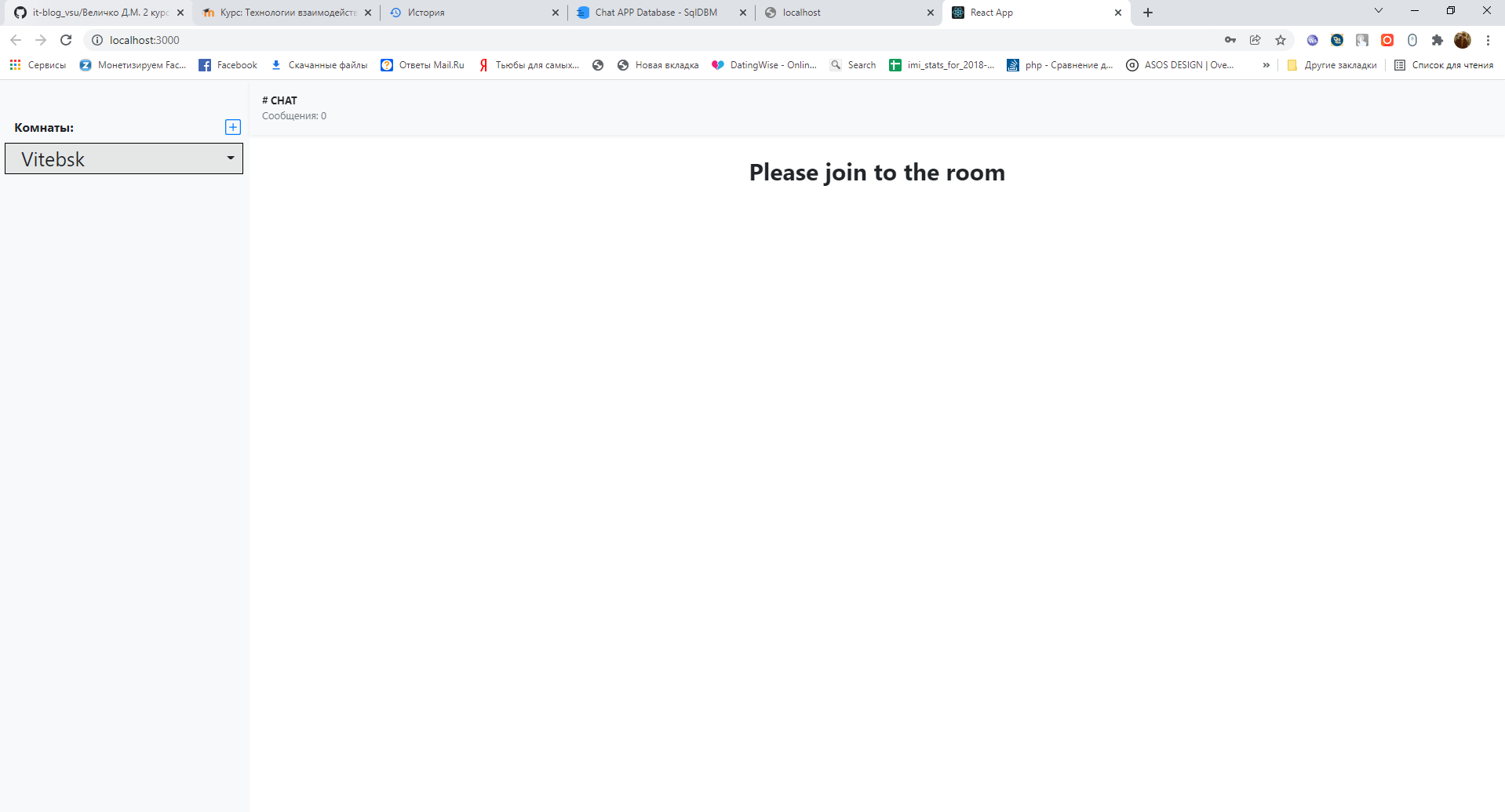


Рисунок 2.4 – Главная страница

Попробуем создать новый чат, для этого нажимаем на иконку возле текста «Комнаты» и видим окно для создания новой комнаты. (рисунок 2.5)

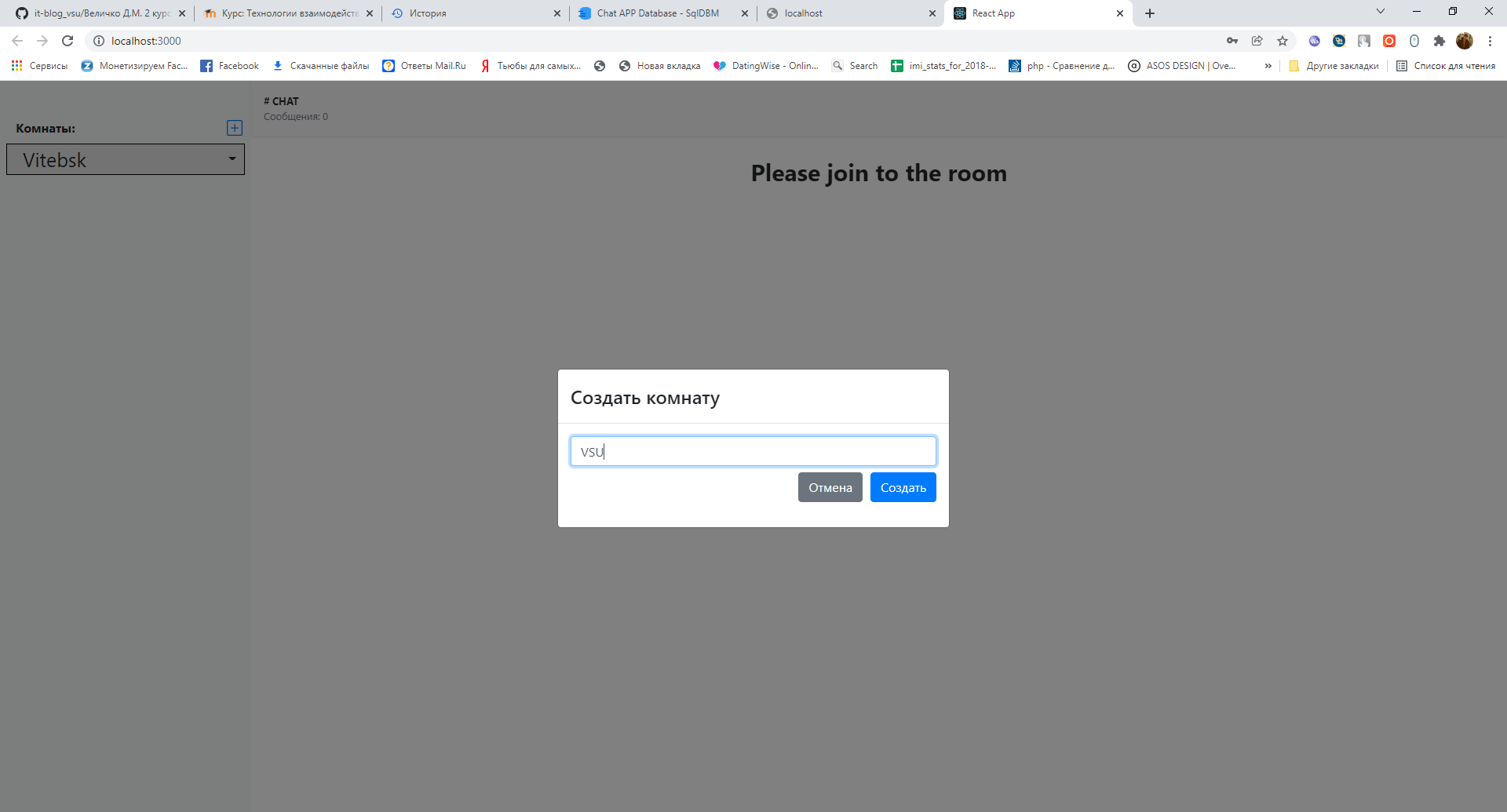


Рисунок 2.5 – Создание новой комнаты

На рисунке 2.6 внизу, вы можете увидеть результат операции (рисунок 2.6).

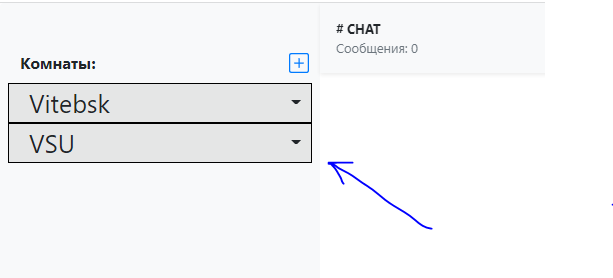


Рисунок 2.6 – Добавленный чат

Так же владелец может по желанию удалить чат.

Присоединимся к чату, нажав на кнопку «Присоединиться». (рисунок 2.7)

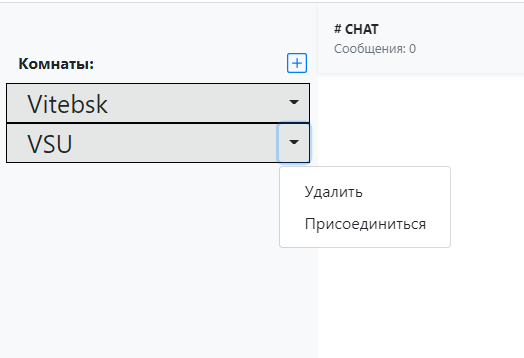


Рисунок 2.7 – Вход в чат

Теперь для проверки войдем с нового клиента, со вкладки «инкогнито» в браузере «Chrome». И попробуем отправить сообщение. Результат показан на рисунке 2.8.

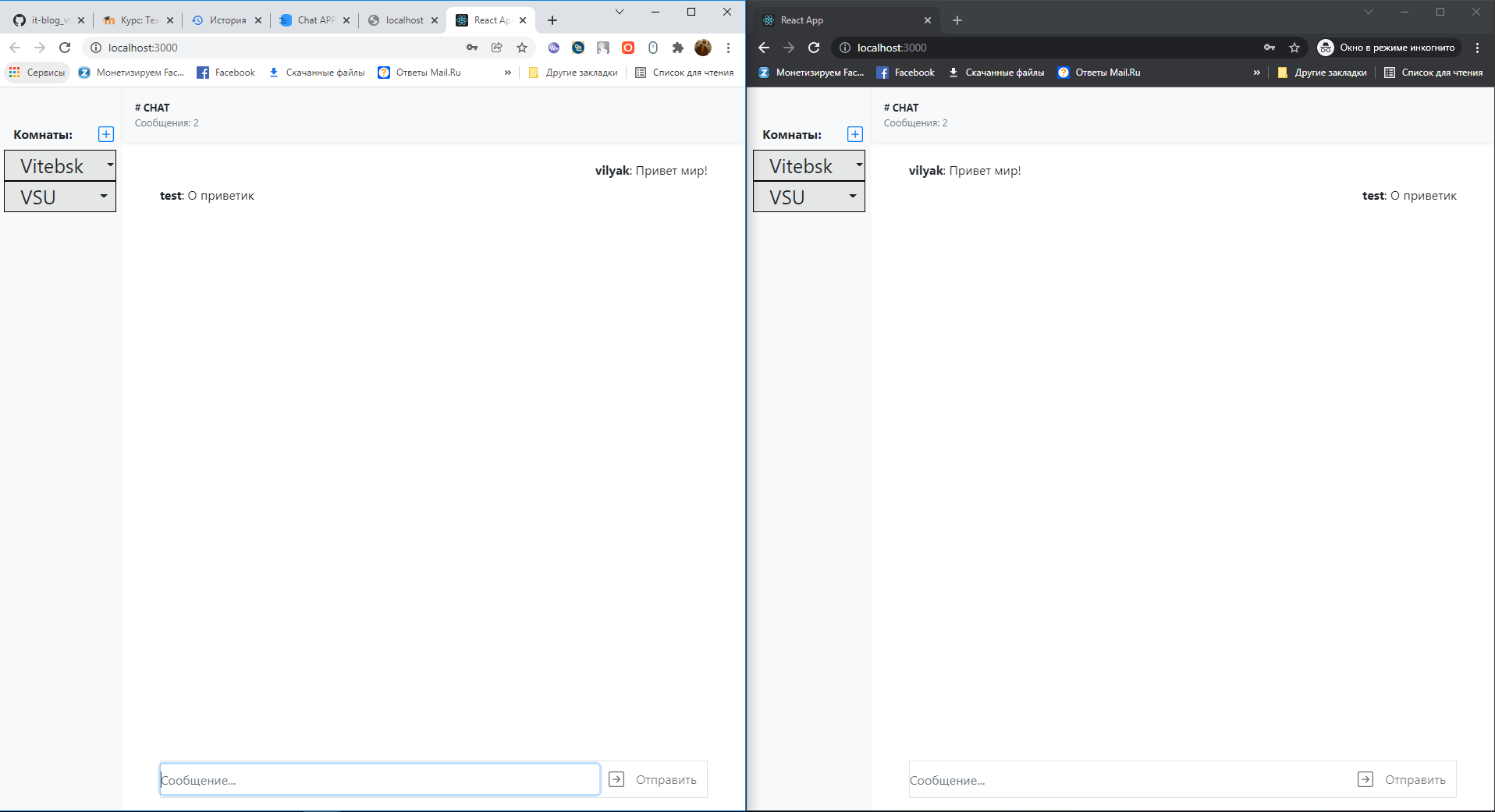


Рисунок 2.7 – Демонстрация работы чата

Видим, что сообщения отправились, и собеседники видят сообщения друг друга. При этом не нужно перезагружать страницу. Так как присутствует соединение через «Websocket» протокол. И сообщения при этом доходят моментально.

# ЗАКЛЮЧЕНИЕ

В ходе курсовой работы был разработан и реализован онлайн чат для общения между пользователями и хранения соответствующей информации в базе данных, та же реализовать вход и регистрацию нового пользователя на языке JavaScript (NodeJS, ReactJS, ExpressJS, Redux), с использованием системы управления базами данных (СУБД) MySQL. Система предоставляет возможность ввод, удаление, хранения, редактирования и просмотр информации. Реализованное приложение будет полезна людям, которые любят общаться в интернете между собой. По окончанию разработки программного продукта были достигнуты следующие цели:

* регистрация пользователя;
* вход в систему;
* отправка сообщения через протокол Websocket;
* вывод информации на экран: текстовой, графической и др.;
* возможность создавать и управлять новыми чатами.

Разработка программы позволила мне укрепить знания в таких технологиях как, NodeJS, ReactJS, ExpressJS, Redux.

Таким образом, цель курсового проектирования была успешно достигнута.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. «ReduxJS wiki» [Электронный ресурс] – Режим доступа - https://ru.wikipedia.org/wiki/Redux/ – Дата доступа: 02.02.2022.
2. «ReactJS wiki» [Электронный ресурс] – Режим доступа - https://ru.wikipedia.org/wiki/React/ – Дата доступа: 02.02.2022.
3. «ExpressJS wiki» [Электронный ресурс] – Режим доступа - https://ru.wikipedia.org/wiki/Express.js/ – Дата доступа: 02.02.2022.
4. «Material UI» [Электронный ресурс] – Режим доступа - https://material-ui.com/ru/– Дата доступа: 02.02.2022.
5. «Документация ReactJS» [Электронный ресурс] – Режим доступа: <https://ru.reactjs.org/docs> – Дата доступа: 02.02.2022.

# ПРИЛОЖЕНИЕ А

ФРАГМЕНТ КОДА (КОМПОНЕНТА), ОТВЕЧАЮЩИЙ ЗА СЕРВЕРНУЮ ЧАСТЬ ОБРАБОТКИ ЗАПРОСОВ

const app = express()

const port = 8080

const wss = new WebSocket.Server({ port: 8081 });

wss.on('connection', function connection(ws) {

ws.on('message', function incoming(data) {

addMessageData(JSON.parse(data), (newMessageData) => {

wss.clients.forEach(function each(client) {

client.send(JSON.stringify(newMessageData));

});

});

});

});

function errorHandler(res: express.Response, action: () => void) {

try {action()}

catch (error) {

res.status(409).send(error);

}

}

function appendCors(res: any){

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

}

app.get(`/${AUTH\_\_PUBLIC\_URL}`, (req, res) => {

errorHandler(res, () => {

auth(req.param('login'), req.param('password'), (data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

})

});

app.get(`/${JOIN\_TO\_ROOM\_URL}`, (req, res) => {

joinToRoom(req.param('userId'), req.param('roomId'), (data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

});

app.get(`/${SIGN\_UP\_URL}`, (req, res) => {

signUp(req.param('login'), req.param('password'), (data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

});

app.get(`/${FETCH\_ROOMS\_URL}`, (req, res) => {

getAllRooms((data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

});

app.get(`/${CREATE\_\_ROOM\_URL}`, (req, res) => {

createRoom(req.param('name'), req.param('userId'), (data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

});

app.get(`/${REMOVE\_\_ROOM\_URL}`, (req, res) => {

removeRoom(req.param('roomId'), req.param('userId'), (data) => {

appendCors(res);

res.send(JSON.stringify(data));

});

});

app.listen(port, () => {

console.log(`Server listening at port : ${port}`);

connectToDatabase((message: string) => {

console.log(message);

});

})

ФРАГМЕНТ КОДА, ОТВЕЧАЮЩИЙ ЗА ЗАПРОСЫ К БАЗЕ ДАННЫМ

import { first } from 'lodash';

import \* as mysql from 'mysql2';

import {databaseConfiguration} from "../config";

import { Message, MessageData, RoomData, UserData } from "../types";

let connection = null;

export function connectToDatabase(callback: (message: string) => void) {

connection = mysql.createPool({...databaseConfiguration, connectionLimit : 10});

callback("Connected to DB!");

}

export function getAllRooms(callback: (data: Array<RoomData>) => void) {

connection.query('SELECT \* FROM rooms', function (error, results, fields) {

if (error) throw error;

callback(results);

});

}

export function createRoom(name: string, userId: string, callback: (data: any) => void) {

connection.query(`INSERT INTO rooms (name, ownerId) VALUES ('${name}', '${userId}')`, function (error) {

if (error) throw error;

connection.query('SELECT id FROM rooms WHERE ownerId = ' + userId, function (error, results, fields) {

if (error) throw error;

const room: {id: number} = first(results);

if (room) {

joinToRoom(userId, room.id.toString(), callback);

}

else console.log('Room is not exist!');

});

});

}

export function joinToRoom(userId: string, roomId: string, callback: (data: any) => void) {

connection.query(`SELECT \* FROM rooms WHERE id = ${roomId}`, function (error, results, fields) {

if (error) throw error;

if (results.length) {

connection.query(`UPDATE users SET currentRoomId = '${roomId}' WHERE id = ${userId}`, function (error, results) {

if (error) throw error;

getAllUsersFromRoom(roomId, (users: Array<UserData>) => {

getAllMessagesFromRoom(roomId, (messages: Array<Message>) => {

callback({status: "OK", data: users, messages, id: roomId});

})

});

});

}

else {

console.log('Room is not exist!');

}

});

}

export function getAllUsersFromRoom(roomId: string, callback: (data: any) => void) {

connection.query('SELECT \* FROM users WHERE currentRoomId = ' + roomId, function (error, results, fields) {

if (error) throw error;

callback(results);

});

}

export function addMessageData({message, roomId, userId, login}: MessageData, callback: (data: any) => void) {

connection.query('SELECT max(localId) FROM messages WHERE roomId = ' + roomId, function (error, results: Array<any>, fields) {

if (error) throw error;

const currentLocalId: number = first(results)?.localId ? Number(first(results)?.localId) : -1;

const localId = currentLocalId + 1;

connection.query(`INSERT INTO messages (localId, roomId, userId, message, login) VALUES (${localId}, ${roomId}, ${userId}, '${message}', '${login}')`, function (error) {

if (error) throw error;

callback({localId, roomId, userId, message, login});

});

});

}

export function getAllMessagesFromRoom(roomId: string, callback: (data: any) => void) {

connection.query('SELECT \* FROM messages WHERE roomId = ' + roomId, function (error, results, fields) {

if (error) throw error;

callback(results);

});

}

export function auth(login: string, password: string, callback: (data: any) => void) {

connection.query(`SELECT password, id FROM users WHERE login = '${login}'`, function (error, result, fields) {

if (error) throw error;

const user: UserData = first(result);

if (user && user.password === password) {

callback({status: "OK", userId: user.id});

}

else {

callback({status: "NO"});

}

});

}

export function removeRoom(roomId: string, userId: string, callback: (data: any) => void) {

connection.query(`DELETE FROM rooms WHERE id = ${roomId} and ownerId = ${userId}`, function (error) {

if (error) throw error;

callback({status: 'OK'});

});

}

export function signUp(login: string, password: string, callback: (data: any) => void) {

connection.query(`SELECT \* FROM users WHERE login = '${login}'`, function (error, users: Array<UserData>) {

if (error) throw error;

if (!users.length) {

connection.query(`INSERT INTO users (login, password, currentRoomId) VALUES ('${login}', '${password}', -1)`, function (error) {

if (error) throw error;

connection.query(`SELECT id FROM users WHERE login = '${login}'`, function (error, result: any) {

if (error) throw error;

callback({status: 'OK', userId: result.id});

});

});

}

else {

callback({status: "NO"});

}

});

}

export function closeConnection() {

connection.end();

}

КОД, ОТВЕЧАЮЩИЙ ЗА ОСНОВНУЮ БИЗНЕС ЛОГИКУ НА КЛИЕНТЕ (Sagas methods)

import {put, takeLatest} from 'redux-saga/effects'

import {getType} from "typesafe-actions";

import {auth, createRoom, deleteRoom, fetchRooms, joinToRoom, signUp} from "./actions";

import {authUser, createUserRoom, deleteUserRoom, getAllRooms, joinToUserRoom, signUpUser} from "./api";

import { JoinRoomResponse, ResponseStatusData, RoomData, UserAuthResponse, UserData } from './types';

export function\* fetchRoomsHandler() {

try {

const rooms: Array<RoomData> = yield getAllRooms();

yield put(fetchRooms.success(rooms));

}

catch (e: any) {

console.error(e.message);

yield put(fetchRooms.failure(e.message));

}

}

export function\* authUserHandler(action: any) {

try {

const data: UserAuthResponse = yield authUser(action.payload);

if (data.status === 'OK') {

const userData: UserData = {

id: data.userId,

currentRoomId: -1,

login: action.payload.login,

password: action.payload.password,

};

yield put(auth.success({...userData, ...data}));

if (action.payload.onData) {

action.payload.onData(data.userId);

}

}

else {

auth.failure(true);

}

}

catch (e: any) {

console.error(e.message);

yield put(auth.failure(true));

}

}

export function\* signUpUserHandler(action: any) {

try {

const data: UserAuthResponse = yield signUpUser(action.payload);

if (data.status === 'OK') {

const userData: UserData = {

id: data.userId,

currentRoomId: -1,

login: action.payload.login,

password: action.payload.password,

};

yield put(signUp.success({...userData, ...data}));

if (action.payload.onData) {

action.payload.onData(data.userId);

}

}

else {

auth.failure(true);

}

}

catch (e: any) {

console.error(e.message);

yield put(auth.failure(true));

}

}

export function\* removeRoomHandler(action: any) {

try {

const data: ResponseStatusData = yield deleteUserRoom(action.payload);

yield put(deleteRoom.success(data));

yield put(fetchRooms.request());

}

catch (e: any) {

console.error(e.message);

yield put(deleteRoom.failure(true));

}

}

export function\* createRoomHandler(action: any) {

try {

const data: JoinRoomResponse = yield createUserRoom(action.payload);

yield put(createRoom.success(data));

yield put(fetchRooms.request());

}

catch (e: any) {

console.error(e.message);

yield put(createRoom.failure(true));

}

}

export function\* joinToRoomHandler(action: any) {

try {

const data: JoinRoomResponse = yield joinToUserRoom(action.payload);

yield put(joinToRoom.success(data));

}

catch (e: any) {

console.error(e.message);

yield put(joinToRoom.failure(true));

}

}

export default function\* rootSaga() {

yield takeLatest(getType(auth.request), authUserHandler);

yield takeLatest(getType(signUp.request), signUpUserHandler);

yield takeLatest(getType(fetchRooms.request), fetchRoomsHandler);

yield takeLatest(getType(deleteRoom.request), removeRoomHandler);

yield takeLatest(getType(createRoom.request), createRoomHandler);

yield takeLatest(getType(joinToRoom.request), joinToRoomHandler);

}

# ПРИЛОЖЕНИЕ Б

Содержание электронного носителя

На электронном носителе расположены следующие файлы и директории:

– Файл «Величко Д.М. – 3 курс – OnlineChat.docx» представляет собой текстовый документ с курсовой работой.

– Директория «Проект» содержит готовый проект курсовой работы.

– Файл «Readme.txt» содержит описание содержания электронного носителя и инструкцию по запуску проекта.