МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Медицинский центр»

Выполнил студент Каминский Дмитрий Викторович

(Ф.И.О.)

Руководитель проекта асс. Нистюк О. А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Нистюк О. А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Нистюк О. А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2022

Содержание

[Введение 3](#_Toc104352479)

[1 Постановка задачи 4](#_Toc104352480)

[3 Проектирование приложения 6](#_Toc104352481)

[3.1 Работа с базой данных 7](#_Toc104352482)

[4 Разработка приложен 11](#_Toc104352483)

[4.1 Проектирование структуры серверной части 11](#_Toc104352484)

[4.2 Проектирование структуры клиентской части 13](#_Toc104352485)

[4.3 Реализация Web-socket 16](#_Toc104352486)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 19](#_Toc104352487)

[6 Руководство по установке и использованию 24](#_Toc104352488)

[6.1 Руководство для неавторизованного пользователя 24](#_Toc104352490)

[6.2 Руководство для аутентифицированного пользователя 24](#_Toc104352491)

[Заключение 27](#_Toc104352492)

[Список литературы 28](#_Toc104352493)

[Приложение А 29](#_Toc104352494)

[Приложение Б 30](#_Toc104352495)

[Приложение В 31](#_Toc104352496)

# Введение

Web-приложение «Медицинский центр» – это приложение, позволяющее совершить запись к врачу, а также отправить на рассмотрение анализы, измеренные дома самостоятельно. Программный продукт позволяет записаться к врачу на одно из свободных мест, получить консультацию, в случае необходимости.

Данное Web-приложение реализовано на платформе Node JS [1], взаимодействует с базой данных MySQL [2]. Серверная часть приложения написана с помощью фреймворка Express JS [3]. Для пользователя доступно React-приложение [4] с асинхронным UI, которое позволяет работать с сервером. Управление сайтом интуитивно, понятно и удобно.

Для выполнения данного программного средства были поставлены следующие задачи:

1. Установление требований к программному средству, описание функциональности.
2. Проектирование, создание логических схем работы программного средства.
3. Программирование, отладка модулей проекта, сборка и комплексная отладка. Описание классов, атрибутов и методов.
4. Проведение тестирования программного средства.
5. Разработка руководства по установке и использованию приложения.

Решение каждой из задач отображено соответственно в главах:

1. Анализ требований к программному средству и разработка функциональных требований.
2. Проектирование приложения.
3. Разработка приложения.
4. Тестирование, проверка работоспособности и анализ полученных результатов.
5. Руководство по установке и использованию.

**1 Постановка задачи**

При первом открытии сайта пользователь не аутентифицирован. Диаграмма вариантов использование неаутентифицированного пользователя отображена на рисунке 2.1.

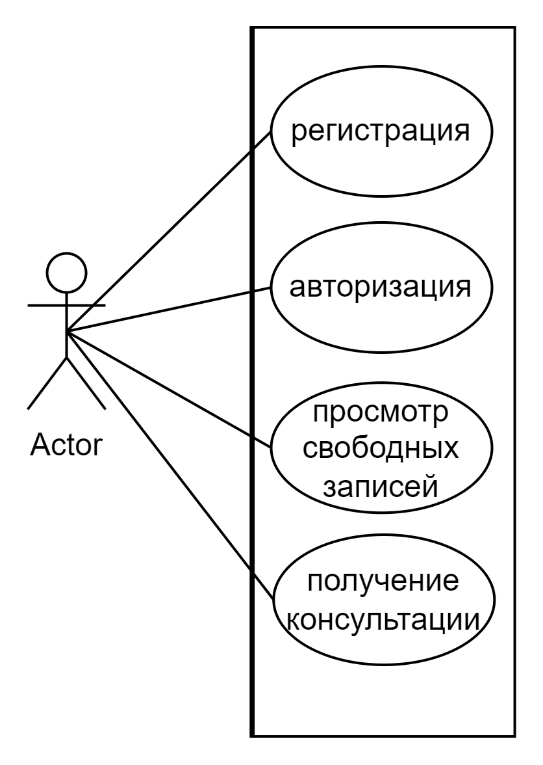


Рисунок 2.1 – Диаграмма использования неаутентифицированного пользователя

Как видно из диаграммы, неаутентифицированный пользователь может зарегистрироваться и авторизоваться. Также пользователю доступен просмотр свободных записей и получение консультации по интересующим вопросам. При попытке записи без авторизации пользователь будет перенаправлен на страницу для авторизации. Остальные действия, недоступные анонимному пользователю скрыты.

Зарегистрировавшись и войдя в аккаунт, пользователь получает токен доступа, в котором записаны его почта, идентификатор. Например, меню удаления пользователя скрыто, пока пользователь не авторизован. Диаграмма вариантов использования приложения авторизованным пользователем представлена рисунке 2.2.



Рисунок 2.2 – Диаграмма использования для аутентифицированного пользователя

# 3 Проектирование приложения

В предыдущей главе были разработаны технические требования к программному средству. Для реализации разрабатываемого проекта была выбрана технология Node JS и база данных MySQL.

Взаимодействие серверной части приложения с реляционной базой данных осуществляется с помощью ORM Sequelize[5]. Помимо этого использование MySQL невозможно без пакета npm mysql2.

Схема базы данных для разрабатываемого программного средства отображена в приложении А. Рассмотрим её подробнее.

Схема отображает таблицы базы данных и связи между ними.

Как видно из схемы, в приложении используется 5 таблиц: USERS, DOCTORS, WORKDAYS, HOME\_ANALYSES и VISITIES.

Таблица USERS содержит данные пользователя для его аутентификации, авторизации и идентификации, а также необходимую информация для врача. Она включает в себя следующие поля:

* ID – уникальный идентификатор записи;
* FULLNAME – ФИО пользователя;
* GENDER– пол пользователя;
* BIRTHDATE – дата рождения пользователя;
* TELENUMBER – номер телефона пользователя;
* EMAIL – адрес электронной почты пользователя;
* LOGIN – логин пользователя;
* PASSWORD – пароль пользователя;
* CREATEDAT – дата создания записи;
* UPDATEDAT – дата изменения записи.

Таблица DOCTORS содержит список пользовательских оценок к фильмам:

* ID – уникальный идентификатор записи;
* FULLNAME – ФИО врача;
* SPECIALITY – специальность врача;
* EMAIL – адрес электронной почты врача;
* LOGIN – логин врача;
* PASSWORD – пароль врача;
* CREATEDAT – дата создания записи;
* UPDATEDAT – дата изменения записи.

Таблица WORKDAYS содержит информацию о фильмах:

* ID – уникальный идентификатор записи;
* DOCTORID – идентификатор врача из таблицы DOCTORS;
* CABINET – поле хранящее значение кабинет врача;
* DAY– поле хранящее дату рабочего дня;
* BEGIN – поле хранящее время начала приема одного человека;
* END – поле хранящее время конца приема одного человека;
* BUSY – поле, показывающее записан ли кто-то на это время;
* CREATEDAT – дата создания записи;
* UPDATEDAT – дата изменения записи.

Таблица HOME\_ANALYSES содержит в себе все данные о чате, а именно:

* ID – уникальный идентификатор записи;
* USERID – идентификатор пользователя из таблицы USERS;
* PULSE – поле хранящее пульс пользователя;
* TEMPERATURE – поле хранящее температуру пользователя;
* BLOOD\_PRESS – поле хранящее артериальное давление пользователя;
* DATE – поле хранящее дату измерения показателей;
* CREATEDAT – дата создания записи;
* UPDATEDAT – дата изменения записи.

Таблица VISITIES содержит в себе все данные о чате, а именно:

* ID – уникальный идентификатор записи;
* DATE – поле хранящее дату записи;
* BEGIN – поле хранящее время начала приема;
* END – поле хранящее время конца приема;
* USERID – идентификатор пользователя из таблицы USERS;
* DOCTORID – идентификатор врача из таблицы DOCTORS;
* CREATEDAT – дата создания записи;
* UPDATEDAT – дата изменения записи.

## 3.1 Работа с базой данных

В разрабатываемом программном средстве существует возможность добавления, изменения и удаления записей в базе данных. Разработано REST API благодаря которому можно взаимодействовать с базой данных не только через web-приложение, но и посредством обычных HTTPS запросов.

Для работы с базой данных используется ORM Sequelize. Это позволяет выстраивать сложную архитектуру БД, выполнять хранимые процедуры. Используя такой подход, можно добиться максимальной производительности приложения за счет снижения времени ответа от базы.

База данных к приложению подключается c помощью параметров, приведенных в листинге 3.3. Из файла конфигурации данные экспортируются. Сюда включается имя и пароль пользователя, название базы данных и используемая бд.

Листинг 3.1. – Подключение к базе данных

module.exports = new Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD,

{dialect: "mysql"})

Продолжение листинга 3.1.

await oracledb.createPool(dbConfig);

oracledb.fetchAsString = [oracledb.CLOB];

export const getConnection = async () => {

  const connection = await oracledb.getConnection("FA\_POOL");

  return connection;

};

На листингах 3.2. – 3.7. представлены примеры создания таблиц и отношений между ними.

Таблица USERS хранит данные о пользователе и необходимую врачу информацию.

const User = sequelize.define('users', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

fullname:{type: DataTypes.STRING(50),allowNull: false},

gender:{type: DataTypes.CHAR(1),allowNull: false},

birthdate:{type: DataTypes.DATEONLY, allowNull: false},

telenumber:{type: DataTypes.STRING(15),allowNull: true},

email:{type: DataTypes.STRING(100), unique: true,allowNull: false},

login:{type: DataTypes.STRING(30), unique: true, allowNull: false},

password:{type: DataTypes.STRING(100),allowNull: false}

})

Листинг 3.2. – Создание таблицы USERS

Таблица DOCTORS хранит данные о враче и необходимую пользователю информацию.

const Doctor = sequelize.define("doctors", {

id:{type: DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

fullname:{type: DataTypes.STRING(50) ,allowNull: false},

speciality:{type: DataTypes.STRING(50) ,allowNull: false},

email:{type: DataTypes.STRING(100) ,allowNull: false},

login:{type: DataTypes.STRING(30) ,allowNull: false},

password:{type: DataTypes.STRING(30) ,allowNull: false},

experience:{type: DataTypes.TINYINT,allowNull: false}

})

Листинг 3.3. – Создание таблицы DOCTORS

Таблица WORKDAYS хранит данные о рабочем дне врача и данные о доступных записях на прием.

const Workday = sequelize.define('workdays', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

cabinet:{type: DataTypes.SMALLINT ,allowNull: false},

day:{type: DataTypes.DATEONLY ,allowNull: false},

begin:{type: DataTypes.TIME ,allowNull: false},

end:{type: DataTypes.TIME ,allowNull: false},

busy:{type: DataTypes.BOOLEAN, allowNull: false, default: false}

})

Листинг 3.4. – Создание таблицы WORKDAYS

Таблица HOME\_ANALYSES хранит данные о анализах пользователя, которые он произвел сам.

const HAnalyse = sequelize.define('home\_analyses', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

pulse:{type: DataTypes.SMALLINT,allowNull: true},

temperature:{type: DataTypes.FLOAT,allowNull: true},

blood\_press:{type: DataTypes.STRING(8),allowNull: true},

date:{type: DataTypes.DATEONLY, allowNull:false}

})

Листинг 3.5. – Создание таблицы HOME\_ANALYSES

Таблица VISITIES хранит данные о забронированных пользователями талонов на поход к врачу.

const Visit = sequelize.define('visities', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

date:{type: DataTypes.DATEONLY ,allowNull: false, timestamps: false},

begin:{type: DataTypes.TIME, allowNull: false},

end:{type: DataTypes.TIME, allowNull: false}

})

Листинг 3.6. – Создание таблицы HOME\_ANALYSES

На листинге 3.7. представлены все связи таблиц, которые использовались при построении данной схемы

User.hasMany(Visit)

Visit.belongsTo(User)

User.hasMany(HAnalyse)

HAnalyse.belongsTo(User)

Doctor.hasMany(Visit)

Visit.belongsTo(Doctor)

Doctor.hasMany(Workday)

Workday.belongsTo(Doctor)

Листинг 3.7. – Связи между таблицами

# 4 Разработка приложен

## 4.1 Проектирование структуры серверной части

На рисунке 4.1 отображена структура разрабатываемого программного средства.

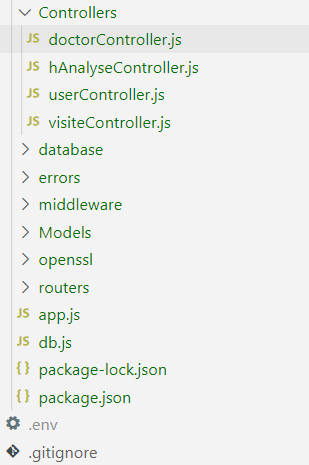


Рисунок 4.1 – Структура проекта сервера

В директории Controllers содержатся контроллеры, которые обрабатывают запросы пользователя на сервер. Пример одного из методов контроллера в Листинге 4.2.

Листинг 4.2. – Пример метода контроллера для получения всех врачей

class VisitController{

async getDoctors(req, res, next){

try{

const doctors = await Doctor.findAll()

if (!doctors) return next(ApiError.badRequest("Can't find any visits"))

res.status(200).json(doctors)

}catch(e){

Продолжение Листинга 4.2

return next(ApiError.internal(e))

}

}

}

module.exports = new VisitController()

В директории errors содержится класс со статическими методами, которые отвечают за создание объектов ошибок. Примеры методов представлены в Листинге 4.3.

static badRequest(message, errors = []){

return new ApiError(400, message, errors)

}

static internal(message){

return new ApiError(500, message)

}

static forbiden(message){

return new ApiError(403, message)

}

static unauthorized(message){

return new ApiError(401, message)

}

Листинг 4.3. – Примеры методов для создания ошибок

Директория middleware, в ней хранятся промежуточные обработчики, которые нужны для проверки JWT токена [6], отлова ошибок и обработки cors [7]. Пример одного из middleware представлен в листинге 4.4.

function cors(req, res, next){

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

res.header("Access-Control-Allow-Methods", "GET, PUT, PATCH, POST, DELETE");

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

next();

}

Листинг 4.4. – Пример middleware обработки cors

Директория Models содержит файл с определенными классами, которые используются для создания таблиц базы данных и их отношений. Примеры описаны в пункте 3.1.

Директория routers содержит роутеры, которые реализуют маршрутизацию в приложении. Пример одного роутрера представлен в листинге 4.5.

const Router = require('express')

const doctorController = require('../Controllers/doctorController')

const router = new Router()

router.get('/', doctorController.getDoctors)

module.exports = router

Листинг 4.5. – Пример userRouter из директории routers

## Проектирование структуры клиентской части

Клиентская часть проекта написана с использованием фреймворка «React». Фреймворк позволяем удобно сочетать язык JavaScript и декларативный подход при написании разметки и стилей. На рисунке 4.5 отображена структура проекта клиентской части.

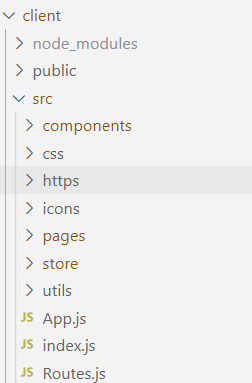


Рисунок 4.6 – Структура клиентской части приложения

Клиентская часть реализует асинхронный подход к реализации пользовательского интерфейса. Асинхронность достигается с помощью отправки асинхронных запросов с помощью библиотеки «axios»

В директории components хранятся все компоненты, необходимые для построения страниц. Пример одного из компонентов показан в Листинге 4.7.

import React, {useContext} from 'react';

import {Switch, Route, Redirect} from 'react-router-dom'

import {publicRoutes} from "../Routes";

import {Context} from "../index";

const AppRouter = () => {

const {user} = useContext(Context)

return (

<Switch>

{publicRoutes.map(({path, Component}) =>

<Route key = {path} path = {path} component={Component} exact/>

)}

</Switch>

);

};

export default AppRouter;

Листинг 4.7. – Пример компонента AppRouter

Директория https содержит методы, реализующие axios для отправки запросов на сервер. Пример одного из них представлен в листинге 4.8.

import {$authHost, $host} from "./index"

export const add = async (date, begin, end, id, idDoc) =>{

const {data} = await $host.post('/api/visit/', {date: date, begin: begin, end: end, id: id, idDoc: idDoc})

}

export const getAll = async (id) =>{

const {data} = await $host.get(`/api/visit/:${id}`)

return data

}

Листинг 4.8. – Пример компонента visitApi

Директория pages содержит основные страницы, которые видит и с которыми взаимодействует пользователь. Пример одного из них представлен в листинге 4.9.

Листинг 4.9. – Пример страницы VisitPage

const VisitPage = observer(() => {

Продолжение листинга 4.9.

const docId = useHistory().location.pathname.split('/')[2]

const {visit} = useContext(Context)

useEffect(()=>{

setTimeout(()=>{getAll(docId).then(data=> {visit.setVisit(data.filter(datum => datum.busy != 1))})

}, 2000)

})

const {doctor} = useContext(Context)

const \_doctor = doctor.\_doctors.find(doc=>doc.id == docId)

return (

<Container fluid>

<Col>

<Row>

<h2>{\_doctor.fullname} {\_doctor.speciality} {visit.\_visites.cabinet}</h2>

</Row>

<Row>

<Col>

<Row>

{visit.\_visites.map(info=>

<VisitItem key={info.id} visit = {info} docId = {docId}/>

)}

</Row>

</Col>

</Row>

</Col>

</Container>

);

})

export default VisitPage;

Директория store содержит основные данные из бд, с которыми имеет дело пользователь при просмотре страниц. Пример одного из них представлен в листинге 4.10.

Листинг 4.10 – Пример класса DoctorStore

export default class DoctorStore{

constructor() {

this.\_doctors = []

this.\_selectedDoc = {}

makeAutoObservable(this)

Продолжение листинга 4.10.

}

setDoctor(doctor){

this.\_doctors =doctor

}

getDoctor(){

return this.\_doctors

}

setSelectDoc(doctor){

this.\_selectedDoc = doctor

}

getSelectDoc(){

return this.\_selectedDoc

}

}

Директория utils содержит файл, в котором хранятся константы путей, которые используются в клиентской части приложения. Пример представлен в листинге 4.11.

export const LOGIN\_DEFAULT\_ROUTE = '/login'

export const LOGIN\_ROUTE = '/login'

export const REGISTER\_ROUTE = '/register'

export const HOMEANALYSE\_ROUTE = '/home\_analyse'

export const VISIT\_ROUTE = '/visit'

export const CONSULTATION\_ROUTE = '/consultation'

export const CHAT\_ROUTE = '/chat'

export const DOCTOR\_ROUTE = '/doctor'

Листинг 4.11. – Пример констант из директории utils

## Реализация Web-socket

Веб-сокеты в данном приложении реализованы с помощью библиотеки Socket.io [8] и используются для реализации консультации пользователя в режиме реального времени.

Реализация серверной части веб-сокеты и часть обрабатываемых событий представлена в листинге 4.12.

const io = require('socket.io')(httpServer, {

cors:{

origin: '\*'

}

})

httpServer.listen(WEBSOCKET\_PORT, ()=>console.log(`Web socket server: http://localhost:${WEBSOCKET\_PORT}`))

const onConnection = (socket)=>{

socket.on('join', ({name, room}, callback)=>{

const {error, user} = addUser({id: socket.id, name:name, room:room})

if(error) return callback(error)

socket.emit('message', {user: 'admin', text: `${user.name}, welcome to the room ${user.room}. Here you can ask any questions and other people will help you`})

socket.broadcast.to(user.room).emit('message', {user: 'admin', text: `${user.name} has joined!`})

socket.join(user.room)

io.to(user.room).emit('roomData', {room: user.room, users: getUsersInRoom(user.room)})

callback()

})

socket.on('sendMessage', (message, callback)=>{

const user = getUser(socket.id)

io.to(user.room).emit('message', {user: user.name, text:message})

io.to(user.room).emit('roomData', {room: user.room, users: getUsersInRoom(user.room)})

callback()

})

socket.on('disconnect', ()=>{

const user = removeUser(socket.id)

if(user)

io.to(user.room).emit('message', {user: 'admin', text: `${user.name} has left`})

})

}

io.on('connection', onConnection)

Листинг 4.12. – Серверная часть web-socket консультации

Клиентская часть web-socket приложения реализована также с использованием React и React-bootstrap. Часть представлена в листинге 4.13.

Листинг 4.13. – Клиентская часть web-socket консультации

useEffect(()=>{

Продолжение листинга 4.13.

const {name, room} = queryString.parse(location.search)

socket = io(ENDPOINT)

setName(name)

setRoom(room)

socket.emit('join', {name, room}, (error)=>{

if(error) alert(error)

})

return ()=>{

socket.emit('disconnect')

socket.off()

}

},[ENDPOINT, location.search])

useEffect(()=>{

socket.on('message', (message)=>{

setMessages([...messages, message])

})

}, [messages])

const sendMessage = (event)=>{

event.preventDefault()

if(message)

socket.emit('sendMessage', message, ()=> setMessage(''))

}

# 5 Тестирование, проверка работоспособности и анализ полученных результатов

В разработанном программном средстве предусматриваются многие негативные сценарии.

Создавая новую учетную запись, предназначенную для помещения в базу данных, при незаполненных обязательных полях на форме появляется сообщение, продемонстрированное на рисунке 5.1. Сообщения появляются одинаковое для всех типов ошибок.

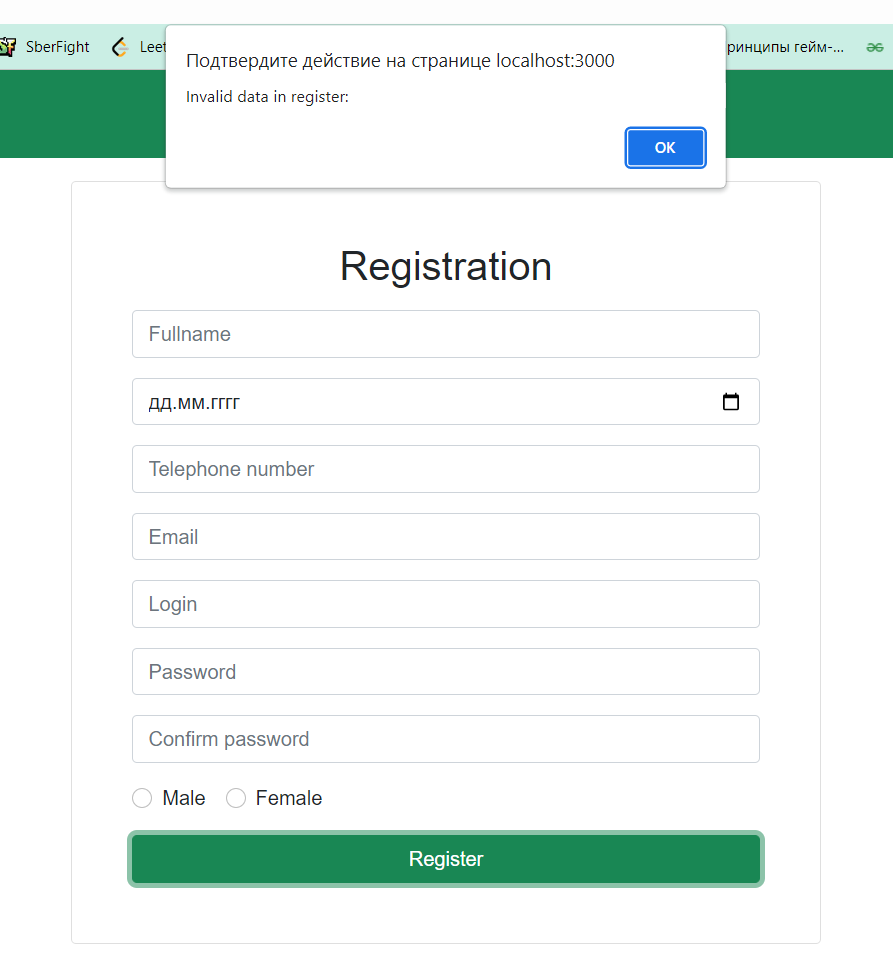


Рисунок 5.1 – Сообщение, возникающее при пустых обязательных полях

Также существует определённый набор правил к вводимым полям:

* поле «email» должно содержать валидный адрес электронной почты­­;
* поле «data» должно содержать только дату
* поле «telephone number» должно быть обязательно не пустым
* поле «login» должно содержать слово длинной больше 7, но меньше 30
* поле «password» должно содержать значение длиной не менее 7 символов и не более 32;
* radiobutton должна быть обязательно нажата;
* поле «confirm password» должно содержать значение, идентичное значению в поле «password».

При попытке зарегистрировать пользователя с логином, который уже имеется в базе данных, выводится сообщение, отображенное на рисунке 5.2.

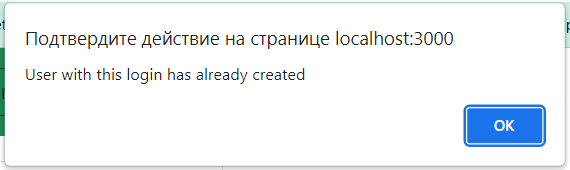


Рисунок 5.2 – Сообщение о существующем пользователе с таким логином

Если пользователь уже зарегистрирован, он может нажать на кнопку «Login». После нажатия откроется форма авторизации в приложение, показанная на рисунке 5.3, где он может ввести своё имя и пароль для получения доступа к дополнительным возможностям приложения.

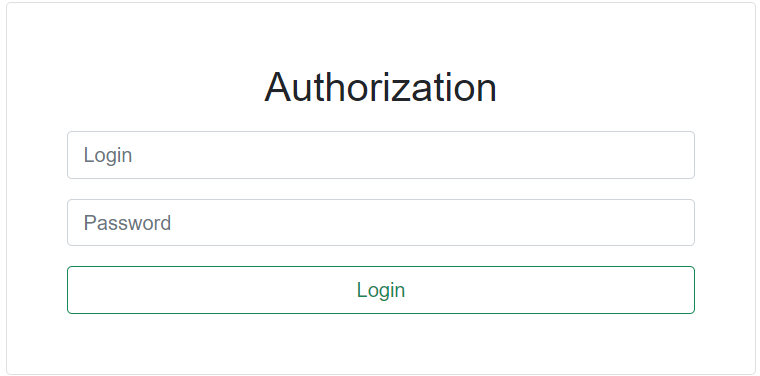


Рисунок 5.3 – Форма авторизации в приложение

Если при входе в систему указываются неверные данные, выводится сообщение об ошибке. Примеры сообщения для неверного логина и пароля продемонстрированы на рисунках 5.4. и 5.5.

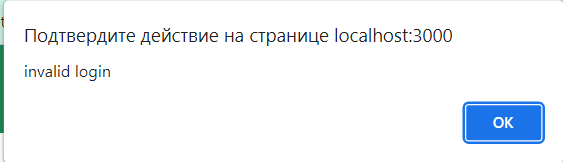


Рисунок 5.4 – Сообщение о неправильно введенном логине пользователя

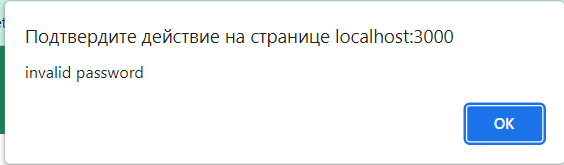


Рисунок 5.4 – Сообщение о неправильно введенном пароле пользователя

При попытке неавторизованного пользователя забронировать талон или отправить анализы, произойдет перенаправление на страницу авторизации с соответствующим сообщением.

Клиентская часть приложения предусматривает роль неавторизованного пользователя и ограничивает некоторый функционал. Например, отсутствует возможность удаления аккаунта или выхода, а вместо меню пользователя кнопки входа и регистрации. Демонстрация представлена на рисунке 5.5 и рисунке 5.6.

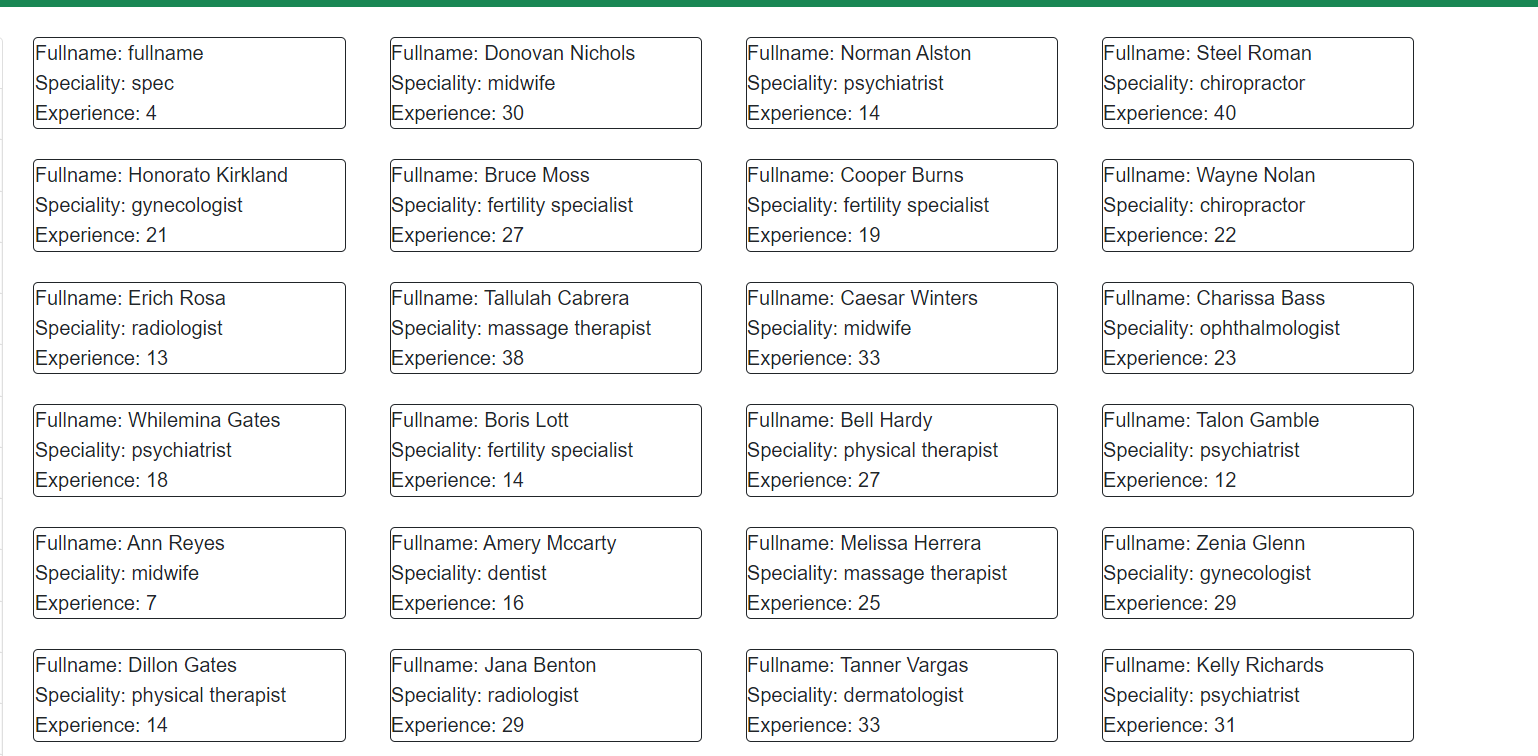


Рисунок 5.7 – Интерфейс неавторизованного пользователя на странице талонов

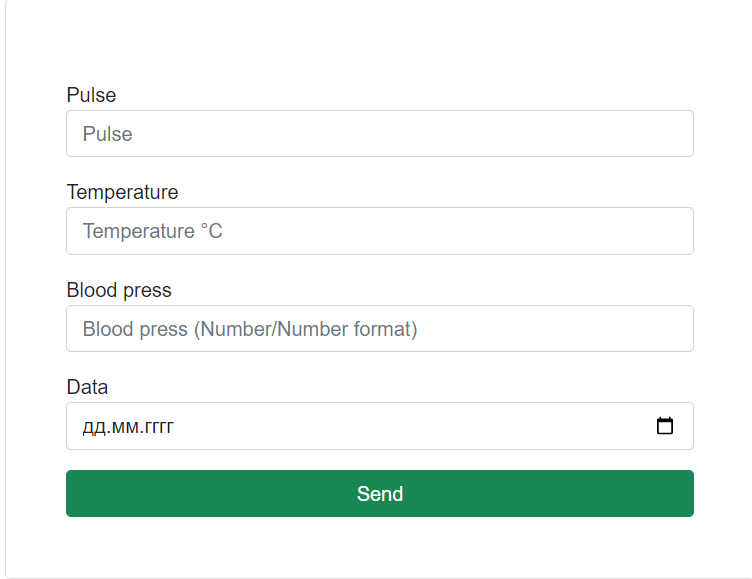


Рисунок 5.8. – Интерфейс неавторизованного пользователя на странице анализов

Кроме клиентской части необходимо проверить и серверную часть, так как она является API, к которому могут посылать запросы другие клиентские приложения. API должен вести себя предсказуемо, обрабатывать всевозможные ошибки во входных моделях и выдавать соответствующие типы ошибок и сообщения.

Для достижения такого поведения каждой конечной точке была создана схема, по которой приложение проверяет входные данные на правильность. Если на этом этапе выясняется, что входная модель не соответствует схеме, сервер генерирует ответ со соответствующим статусом и ошибкой с указанием на поле в объекте, которое не прошло проверку. Данный сценарий описан на рисунке 5.8.

Здесь сторонним пользовательским приложением выступает программа «Postman». Приложение успешно обрабатывает запросы, как от разработанной в рамках курсового проекта клиентской части, так и от стороннего приложения, которое может посылать запросы. Отсюда следует, что разработанный сервер является независимым от клиентской части.

Пройдя валидацию по схеме, необходимо убедиться, что данные, которые находятся во входной модели не нарушают целостности БД. За проверки такого типа отвечает бизнес-слой приложения.

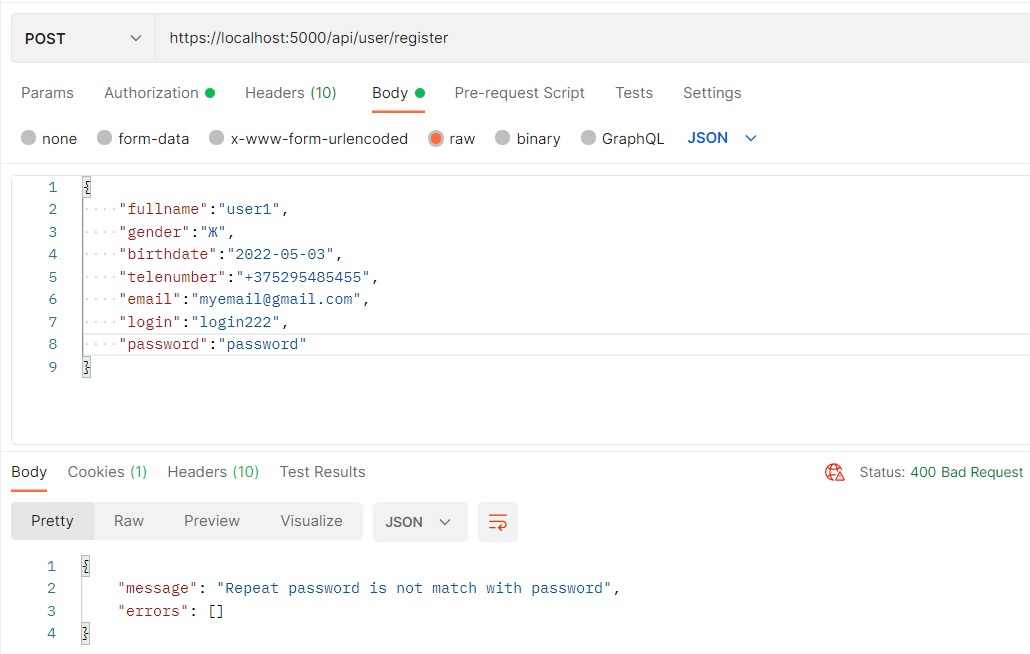


Рисунок 5.8 – Ошибка при отправке невалидной модели при регистрации

# 6 Руководство по установке и использованию

Для корректной работы приложения необходимо наличие компьютера со всеми необходимыми для его работы компонентами (процессором, видеокартой, оперативной памятью, монитором и т.д.), браузер-клиент, а также подключение к сети Internet.

Фактически, установка для разработанного в данном курсовом проекте приложения не требуется. Если необходимо запустить приложение в локальной сети, нужно установить конфигурацию приложения, в частности скачать все используемые пакеты из npm. Так как сама база данных находится в облаке ее конфигурация уже произведена и не требует каких-либо действий при разворачивании сервера на другой машине.

Чтобы иметь возможность пользоваться основным функционалом приложения, необходимо создать аккаунт.

При входе в аккаунт, необходимо ввести данные, указанные при регистрации.



## Руководство для неавторизованного пользователя

В разработанном веб-приложении пользователь является неавторизованным, если токен, хранящийся в localstorage браузера, не является действующим, или же если токена вообще нет.

Функционал неавторизованного пользователя представляет собой авторизацию, регистрацию, просмотр доступных врачей и доступных талонов.

Для создания аккаунта необходимо ввести ФИО, дату рождения, номер телефона, адрес электронной почты, пароль, а также его подтверждение, и выбрать соответствующий пол при этом логин должен быть уникален и быть не меньше 7 символов и не больше 30, адрес почты должны быть уникальными быть не более 100 символов, а пароль должен быть не короче 7 символов и меньше 32.

Пользователь также может войти в созданный аккаунт, заполнив поля на форме аутентификации. Чтобы аутентификация прошла успешно, пользователю необходимо указать логин и пароль, которые были указаны при регистрации.

Если неавторизованный пользователь совершит попытку выполнить действия или открыть страницы, которые доступны только авторизованным пользователям, запрос будет перенаправлен на страницу авторизации.

## Руководство для аутентифицированного пользователя

После аутентификации, у пользователя появляется возможность записываться к врачу.

Чтобы записаться к врачу необходимо зайти на страницу Tickets, выбрать нужного врача, и нажать на комфортное время, страницы показаны на рисунках 6.1., 6.2, соответственно.

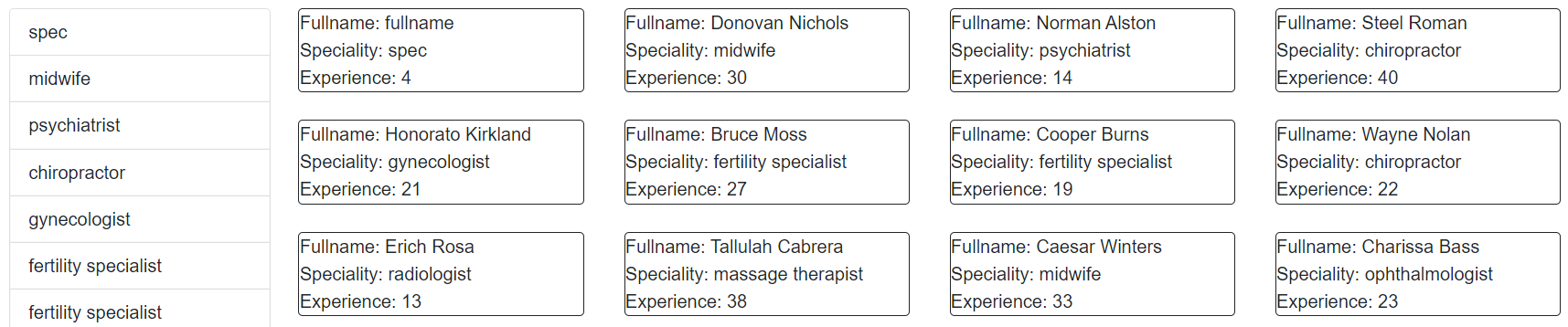


Рисунок 6.1 – Страница Tickets

В верхней части страницы указаны ФИО врача и его специальность

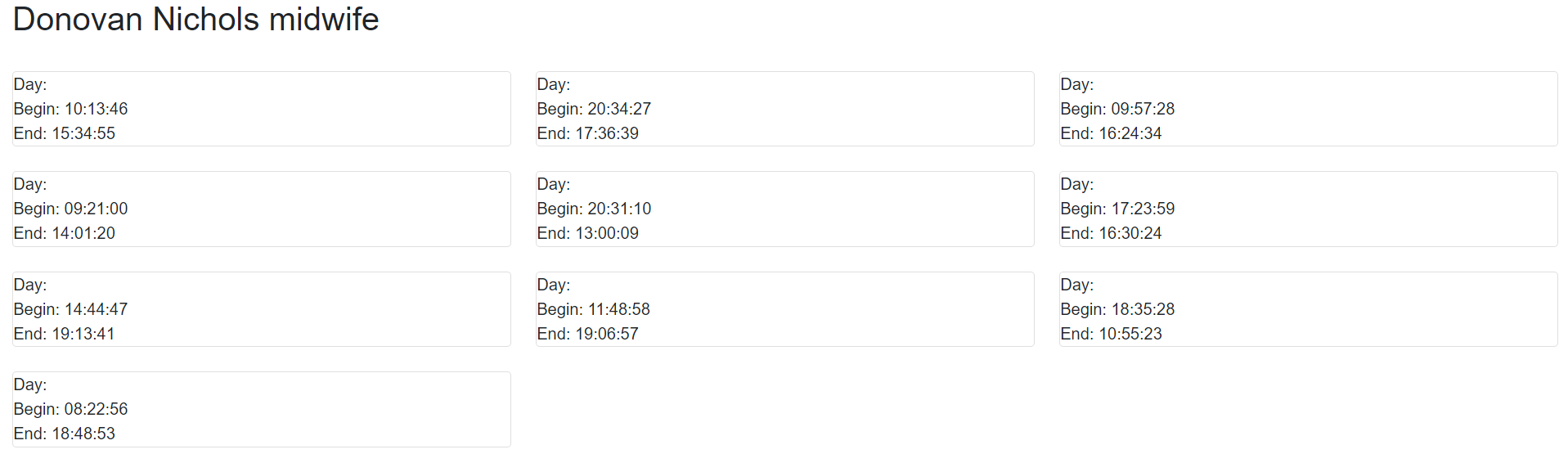


Рисунок 6.2. – Страница Doctor

Авторизованный пользователь может отправлять и изменять свои анализы. Для этого используется страница “Home analyses”, показанная на рисунке 6.3.

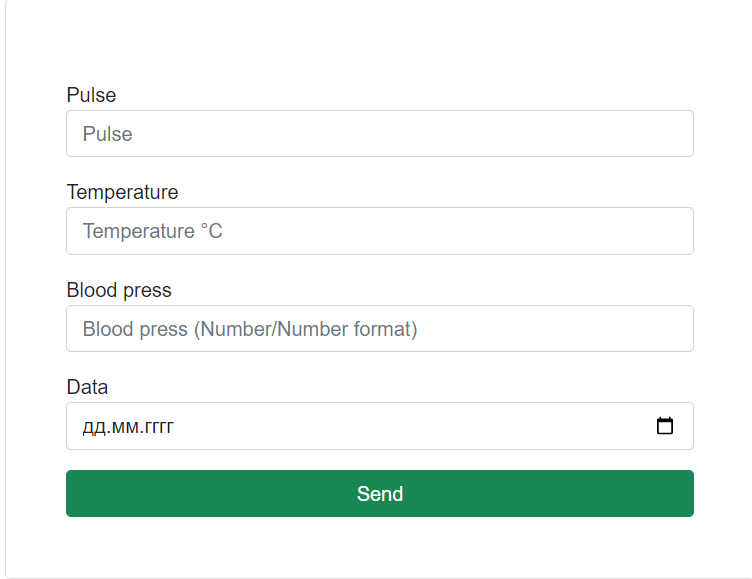


Рисунок 6.3 – Страница с добавлением/изменением анализов

Обязательным к заполнению является поле с датой, остальные могут быть добавлены при дальнейших изменениях. Пульс должен находится в диапазоне между 40 и 250, температура может быть больше 33 и меньше 44. А кровяное давление должно быть в соответствии с шаблоном a/b, где a и b являются числами, начиная от 50 и до 250. В случае неверных значений, будет выведена ошибка и перенаправление в позицию логина.

# Заключение

В результате курсовой работы было разработано Web-приложение «Медицинский центр». Основой приложения стал язык JavaScript. Серверная часть написана с помощью фреймворка Express, а клиентская с помощью фреймворка React. Обе технологии поддерживают асинхронное выполнение операций, что позволяет добиться высокой производительности приложения. В качестве базы данных выступила база от компании Oracle.

Программное средство выполняет следующие функции:

* имеет интуитивно понятное и простое управление;
* обращается к базе данных, успешно извлекает и вносит новые данные, осуществляет поиск по базе и автоматическую запись некоторых полей;
* имеет различный функционал в зависимости от учётной записи.
* быстро обрабатывает запросы (не более 5 секунд);
* функционирует по протоколам https и ws;
* выглядит аккуратно, не содержит лишнюю информацию.

Разработанное программное средство удовлетворяет всем требованиям технического задания, а также реализует несколько дополнительных функций.

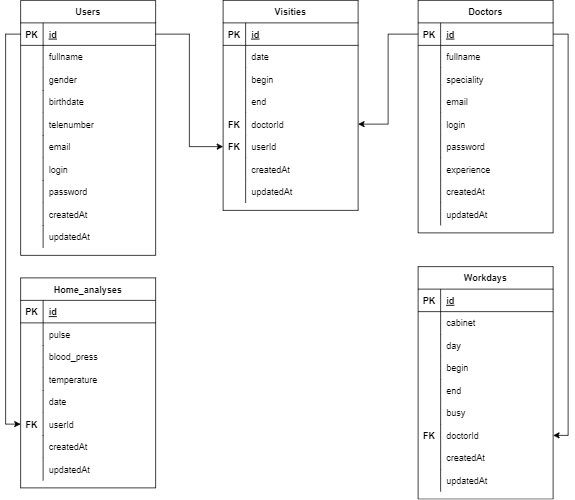
Приложение поддерживает асинхронный интерфейс и обработку пользовательских событий в режиме реального времени.

Приложение было успешно протестировано. В результате проведённых тестов ошибок обнаружено не было. Данный программный продукт является завершённым и готов к использованию.

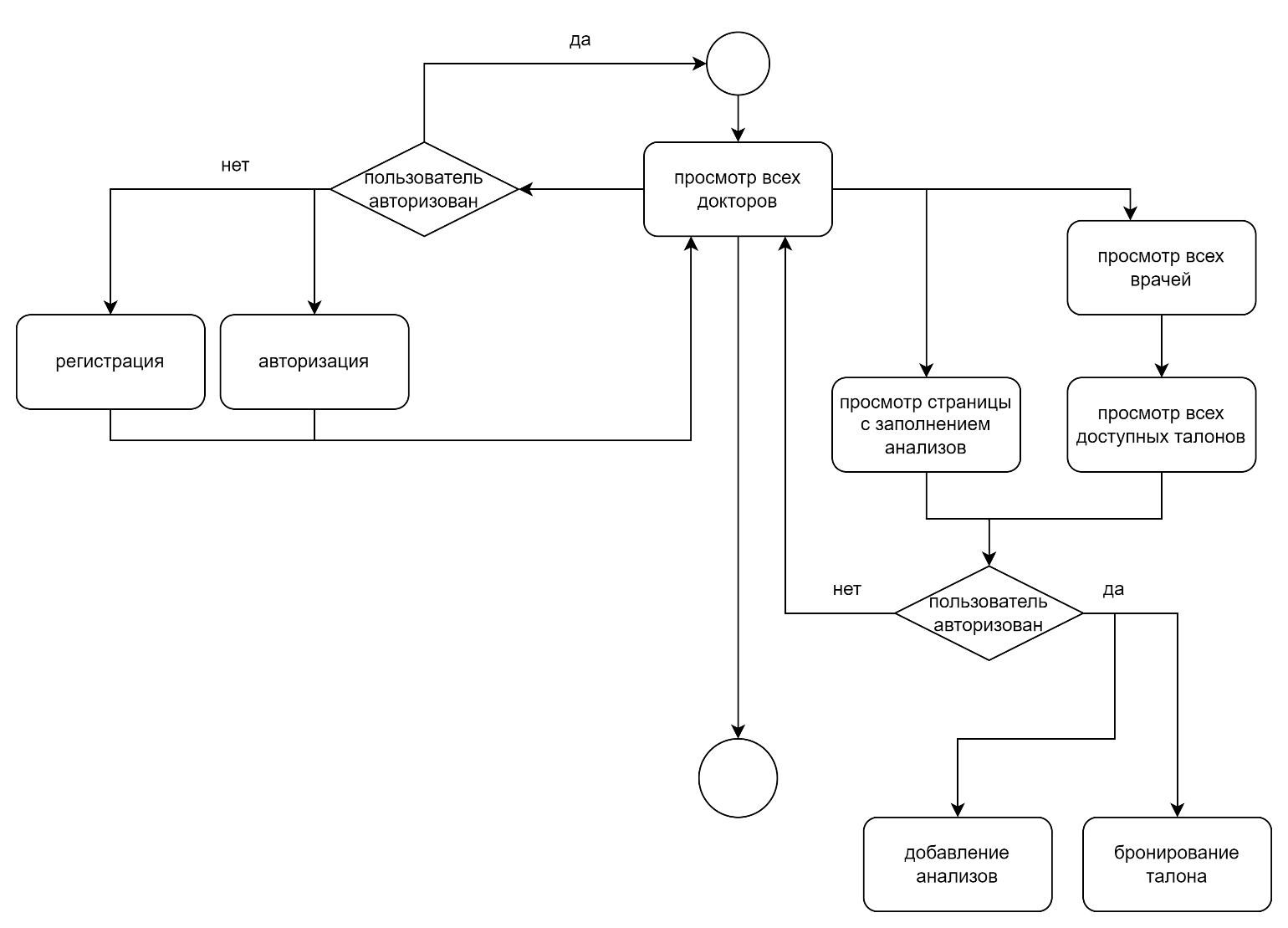
# Список литературы

1. About Node.js [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/about/> – Дата доступа:09.04.2022.
2. СУБД MySQL [Электронный ресурс] – Режим доступа: https://docs.oracle.com/en-us/iaas/mysql-database/doc/getting-started.html – Дата доступа: 10.03.2022;
3. Express [Электронный ресурс] – Режим доступа: <https://expressjs.com/ru/> – Дата доступа: 02.04.2022.
4. React documentation [Электронный ресурс] – Режим доступа: https://ru.reactjs.org/docs/getting-started.html – Дата доступа: 15.04.2022.
5. Sequelize documentation [Электронный ресурс] – Режим доступа: https://sequelize.org/docs/v6/ – Дата доступа: 18.04.2022.
6. Jwt documentation [Электронный ресурс] – Режим доступа: https://nodejsdev.ru/doc/jwt/ – Дата доступа: 27.04.2022.
7. Cors documentation [Электронный ресурс] – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS – Дата доступа: 15.05.2022.
8. Socket.io documentation [Электронный ресурс] – Режим доступа: https://socket.io/docs/v4/ – Дата доступа: 10.05.2022.

# Приложение А



# Приложение Б



# Приложение В

require('dotenv').config();

const express = require("express")

const jwt = require('jsonwebtoken')

const fs = require('fs')

const app = express()

const options = {

passphrase: "qwerty",

key: fs.readFileSync("openssl/CA.key").toString(),

cert: fs.readFileSync("openssl/CA.crt").toString()

}

const httpsServer = require('https')

const httpServer = require('http').createServer(app)

const io = require('socket.io')(httpServer, {

cors:{

origin: '\*'

}

})

const {addUser, getUser, removeUser, getUsersInRoom} = require('./users');

const sequelize = require('./db')

const cookieParser = require('cookie-parser')

const errorHandler = require('./middleware/errorApiMiddleware')

const cors = require('./middleware/cors')

const router = require('./routers/index')

app.use(cors)

app.use(express.json())

app.use((req, res, next)=>{

if(req.headers.authorization){

jwt.verify(req.headers.authorization.split(' ')[1], process.env.JWT\_ACCESS\_KEY, (err, payload)=>{

if(err) next()

else if(payload){

req.payload = payload

next()

}

})

}else next()

})

app.use('/api', router)

app.use(cookieParser())

app.use(errorHandler)

const onConnection = (socket)=>{

socket.on('join', ({name, room}, callback)=>{

const {error, user} = addUser({id: socket.id, name:name, room:room})

if(error) return callback(error)

socket.emit('message', {user: 'admin', text: `${user.name}, welcome to the room ${user.room}. Here you can ask any questions and other people will help you`})

socket.broadcast.to(user.room).emit('message', {user: 'admin', text: `${user.name} has joined!`})

socket.join(user.room)

io.to(user.room).emit('roomData', {room: user.room, users: getUsersInRoom(user.room)})

callback()

})

socket.on('sendMessage', (message, callback)=>{

const user = getUser(socket.id)

io.to(user.room).emit('message', {user: user.name, text:message})

io.to(user.room).emit('roomData', {room: user.room, users: getUsersInRoom(user.room)})

callback()

})

socket.on('disconnect', ()=>{

const user = removeUser(socket.id)

if(user)

io.to(user.room).emit('message', {user: 'admin', text: `${user.name} has left`})

})

}

io.on('connection', onConnection)

const WEBSOCKET\_PORT = process.env.WEBSOCKET\_PORT || 5001

const PORT = process.env.PORT || 5000

const start = async ()=>{

try{

await sequelize.authenticate()

await sequelize.sync()

httpServer.listen(WEBSOCKET\_PORT, ()=>console.log(`Web socket server: http://localhost:${WEBSOCKET\_PORT}`))

httpsServer.createServer(options, app).listen(PORT, ()=> console.log(`Common server: https://localhost:${PORT}/api`))

}catch(e){

console.log(e)

}

}

start();

const Router = require('express')

const router = new Router()

const userRouter = require('./userRouter')

const visitRouter = require('./visitRouter')

const doctorRouter = require('./doctorRouter')

const hAnalyseRouter = require('./hAnalyseRouter')

router.use('/user', userRouter)

router.use('/visit', visitRouter)

router.use('/hAnalyse', hAnalyseRouter)

router.use('/doctor', doctorRouter)

module.exports = router

const Router = require('express')

const doctorController = require('../Controllers/doctorController')

const router = new Router()

router.get('/', doctorController.getDoctors)

module.exports = router

const Router = require('express')

const {body} = require('express-validator')

const hAnalyseController = require('../Controllers/hAnalyseController')

const router = new Router()

router.get('/', hAnalyseController.getPage)

router.post('/', hAnalyseController.create)

module.exports = router

const Router = require('express')

const {body} = require('express-validator')

const userController = require('../Controllers/userController')

const authMiddleware = require('../middleware/authMiddleware')

const router = new Router()

router.post('/login',userController.login)

router.post('/register',

body('fullname').isLength({max:50}),

body('gender').isLength({min: 1, max: 1}),

body('birthdate').isDate(),

body('telenumber').not().isEmpty(),

body('email').isEmail(),

body('login').isLength({min: 7, max: 30}),

body('password').isLength({min:7, max:32}),

userController.register)

router.delete('/deleteUser/:id', userController.destroyUser)

module.exports = router

const Router = require('express')

const visitController = require('../Controllers/visiteController')

const router = new Router()

router.get('/', visitController.getPage)

router.get('/:id', visitController.getAllVisits)

router.get('/:id/:userId', visitController.getAllUserVisits)

router.post('/', visitController.create)

module.exports = router

const sequelize = require('../db');

const {DataTypes} = require('sequelize');

const Doctor = sequelize.define("doctors", {

id:{type: DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

fullname:{type: DataTypes.STRING(50) ,allowNull: false},

speciality:{type: DataTypes.STRING(50) ,allowNull: false},

email:{type: DataTypes.STRING(100) ,allowNull: false},

login:{type: DataTypes.STRING(30) ,allowNull: false},

password:{type: DataTypes.STRING(30) ,allowNull: false},

experience:{type: DataTypes.TINYINT,allowNull: false}

})

const HAnalyse = sequelize.define('home\_analyses', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

pulse:{type: DataTypes.SMALLINT,allowNull: true},

temperature:{type: DataTypes.FLOAT,allowNull: true},

blood\_press:{type: DataTypes.STRING(8),allowNull: true},

date:{type: DataTypes.DATEONLY, allowNull:false}

})

const User = sequelize.define('users', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

fullname:{type: DataTypes.STRING(50),allowNull: false},

gender:{type: DataTypes.CHAR(1),allowNull: false},

birthdate:{type: DataTypes.DATEONLY, allowNull: false},

telenumber:{type: DataTypes.STRING(15),allowNull: true},

email:{type: DataTypes.STRING(100), unique: true,allowNull: false},

login:{type: DataTypes.STRING(30), unique: true, allowNull: false},

password:{type: DataTypes.STRING(100),allowNull: false}

})

const Visit = sequelize.define('visities', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

date:{type: DataTypes.DATEONLY ,allowNull: false, timestamps: false},

begin:{type: DataTypes.TIME, allowNull: false},

end:{type: DataTypes.TIME, allowNull: false}

})

const Workday = sequelize.define('workdays', {

id:{type:DataTypes.INTEGER, primaryKey:true, autoIncrement:true, allowNull:false},

cabinet:{type: DataTypes.SMALLINT ,allowNull: false},

day:{type: DataTypes.DATEONLY ,allowNull: false},

begin:{type: DataTypes.TIME ,allowNull: false},

end:{type: DataTypes.TIME ,allowNull: false},

busy:{type: DataTypes.BOOLEAN, allowNull: false, default: false}

})

User.hasMany(Visit)

Visit.belongsTo(User)

User.hasMany(HAnalyse)

HAnalyse.belongsTo(User)

Doctor.hasMany(Visit)

Visit.belongsTo(Doctor)

Doctor.hasMany(Workday)

Workday.belongsTo(Doctor)

module.exports = {Doctor, HAnalyse, User, Visit, Workday}

const ApiError = require("../errors/ApiError")

const jwt = require('jsonwebtoken')

module.exports = function (req, res, next){

if(req.method === 'OPTIONS')

next()

try{

const token = req.headers.authorization.split(' ')[1]

if(!token) return next(ApiError.unauthorized('Не авторизован'))

const decoded = jwt.verify(token, process.env.JWT\_ACCESS\_KEY)

req.user = decoded

next()

}catch(e){

return next(ApiError.unauthorized('Не авторизован'))

}

}

function cors(req, res, next){

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

res.header("Access-Control-Allow-Methods", "GET, PUT, PATCH, POST, DELETE");

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

next();

}

module.exports = cors

const ApiError = require('../errors/ApiError')

module.exports = function (err, req, res, next){

if(err instanceof ApiError)

return res.status(err.status).json({message:err.message, errors: err.errors})

res.status(500).json({message: "unusual error"})

}

class ApiError extends Error{

status;

errors;

constructor(status, message, errors = []) {

super(message);

this.status = status

this.errors = errors

}

static badRequest(message, errors = []){

return new ApiError(400, message, errors)

}

static internal(message){

return new ApiError(500, message)

}

static forbiden(message){

return new ApiError(403, message)

}

static unauthorized(message){

return new ApiError(401, message)

}

}

module.exports = ApiError

const {Visit, Workday} = require('../Models/models')

const {Desiase} = require('../Models/models')

const ApiError = require('../errors/ApiError')

class VisitController{

getPage(req, res){

}

async getAllUserVisits(req, res, next){

try{

const {id, userId} = req.params

const visit = await Visit.findAll({where:{userId: id}})

if (!visit) return next(ApiError.badRequest("Can't find any visits"))

res.status(200).json(visit)

}catch(e){

return next(ApiError.internal(e))

}

}

async getAllVisits(req, res, next){

try{

const id = req.params.id[1]

const workdays = await Workday.findAll({where:{doctorId: id}})

if (!workdays) return next(ApiError.badRequest("Can't find any visits"))

res.status(200).json(workdays)

}catch(e){

return next(ApiError.internal(e))

}

}

async create(req, res, next){

try {

const {id, idDoc, date, begin, end} = req.body

if(!id) return next(ApiError.unauthorized('Login before book'))

if (!idDoc || !date ||!begin || !end) return next(ApiError.badRequest('Input doctor or date/time'))

const visit = await Visit.create({

userId: id,

doctorId: idDoc,

date: date,

begin: begin,

end: end

})

if (!visit) return next(ApiError.badRequest("Can't create this visit, try again with other data"))

let workday = await Workday.findOne({ where: {

doctorId: idDoc

}

})

if(!workday) return next(ApiError.forbiden('Doctor have no this time'))

workday = await Workday.update(

{

cabinet: workday.cabinet,

day: workday.day,

begin: workday.begin,

end: workday.end,

busy: 1,

createdAt: workday.createdAt,

updatedAt: workday.updatedAt

},

{

where: {id: workday.id}

}

)

if(!workday) return next(ApiError.forbiden('Doctor have no this time'))

res.status(200).json(visit)

}catch(e){

return next(ApiError.internal(e))

}

}

}

module.exports = new VisitController()

const bcrypt = require('bcrypt')

const jwt = require('jsonwebtoken')

const {validationResult} = require('express-validator')

const {User} = require('../Models/models')

const ApiError = require("../errors/ApiError")

const userService = require('../')

class UserController {

async login(req, res, next) {

try {

const user = await User.findOne({

where: {

login: req.body.login

}

});

if (!user) return next(ApiError.internal('invalid login'))

if (! await bcrypt.compare(req.body.password, user.password)) return next(ApiError.internal('invalid password'))

const accessToken = jwt.sign({id: user.id, login:user.login}, process.env.JWT\_ACCESS\_KEY, {expiresIn:'24h'})

const refreshToken = jwt.sign({ip: req.socket.remoteAddress}, process.env.JWT\_REFRESH\_KEY, {expiresIn: '30d'})

res.cookie('refreshToken', refreshToken, {maxAge: 30\*24\*60\*60\*1000, httpOnly:true})

return res.status(200).json({accessToken, refreshToken, user})

}catch(e){

return next(ApiError.internal(e))

}

}

async register(req, res, next) {

try {

const error = validationResult(req)

const {fullname, gender, birthdate, telenumber, email, login, password, rep\_pass} = req.body

if(!error.isEmpty()){

return next(ApiError.badRequest('Invalid data in register: ', error.array()))

}

const candidate = await User.findOne({where:{login: login}})

if(candidate) return next(ApiError.internal('User with this login has already created'))

if(password != rep\_pass) return next(ApiError.badRequest('Repeat password is not match with password'))

const hashPassword = await bcrypt.hash(password, 4)

const user = await User.create({

fullname: fullname,

gender: gender,

birthdate: birthdate,

telenumber: telenumber,

email: email,

login: login,

password: hashPassword

})

return res.json(user)

}catch (e){

return next(ApiError.internal(e))

}

}

async destroyUser(req, res, next){

try{

const id = req.params.id.split(':')[1]

if(!id) return next(ApiError.badRequest('Id is not given'))

const user = await User.destroy({

where:{id: id}

})

if(!user) return next(ApiError.internal("Can find this user"))

return res.json(user)

}catch(e){

return next(ApiError.internal(e))

}

}

}

module.exports = new UserController()

const {HAnalyse} = require('../Models/models')

const ApiError = require('../errors/ApiError')

const {toInt} = require("validator");

const DEFAULT\_VALUE = 0

class HAnalyseController{

getPage(req, res){

}

async create(req, res, next){

try {

const {id, pulse, temperature, blood\_press = '', date} = req.body

if(!id) return next(ApiError.unauthorized('Login before send analyses'))

if (!date) return next(ApiError.badRequest('Input date to analyse'))

let findValue = await HAnalyse.findOne({where:{userId: id, date:date}})

AnalyseValidation(pulse, temperature, blood\_press)

let hAnalyse

if(findValue) {

findValue = setUser(pulse, temperature, blood\_press, findValue)

hAnalyse = await HAnalyse.update({

pulse: findValue.pulse,

temperature: findValue.temperature,

blood\_press: findValue.blood\_press,

}, {

where: {userId: id}

})

}

else {

hAnalyse = await HAnalyse.create({

pulse: pulse,

temperature: temperature,

blood\_press: blood\_press,

date: date,

userId: id

})

}

if (!hAnalyse) return next(ApiError.internal("Can't add this analyse, check validation of data"))

res.status(200).json(hAnalyse)

}catch(e){

return next(ApiError.internal(e))

}

}

}

setUser = (pulse, temperature, blood\_press, findValue)=>{

if(pulse !== '') findValue.pulse = pulse

if(temperature !== '') findValue.temperature = temperature

if(blood\_press !== '') findValue.blood\_press = blood\_press

return findValue

}

AnalyseValidation = (pulse, temperature, blood\_press)=>{

const MAX\_PULSE = 250, MIN\_PULSE = 40, MAX\_TEMP = 44, MIN\_TEMP = 33

const regexBloodPress = /\d{2,3}[/]\d{2,3}/gm

if(blood\_press !== '' && !regexBloodPress.test(blood\_press)) throw "Invalid value of blood\_press"

const checkValue = toInt(blood\_press.split('/')[1]) > 250 || toInt(blood\_press.split('/')[1]) < 50 || toInt(blood\_press.split('/')[2]) > 250 || toInt(blood\_press.split('/')[2]) < 250

if(checkValue) throw "Invalid value of blood\_press"

if(pulse >= MAX\_PULSE || pulse <= MIN\_PULSE)

if (pulse !== DEFAULT\_VALUE) throw "Invalid value of pulse"

if(temperature > MAX\_TEMP || temperature < MIN\_TEMP)

if(temperature !== DEFAULT\_VALUE) throw "Invalid value of temperature"

}

module.exports = new HAnalyseController()

const {Doctor} = require('../Models/models')

const ApiError = require('../errors/ApiError')

class VisitController{

async getDoctors(req, res, next){

try{

const doctors = await Doctor.findAll()

if (!doctors) return next(ApiError.badRequest("Can't find any visits"))

res.status(200).json(doctors)

}catch(e){

return next(ApiError.internal(e))

}

}

}

module.exports = new VisitController()

const {Sequelize} = require('sequelize');

module.exports = new Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD,

{dialect: "mysql"})

const users = []

const addUser = ({id, name, room})=>{

name = name.trim().toLowerCase()

room = room.trim().toLowerCase()

const existingUser = users.find((user)=> user.room === room && user.name === name)

if(existingUser)

return {error: 'Username is taken'}

const user = {id, name, room}

users.push(user)

return {user}

}

const removeUser = (id)=>{

const index = users.findIndex((user)=> user.id === id)

if(index !== -1)

return users.splice(index, 1)[0]

}

const getUser = (id)=> users.find((user)=> user.id === id)

const getUsersInRoom = (room)=> users.filter((user)=> user.room === room)

module.exports = {addUser, removeUser, getUser, getUsersInRoom}

import React, {useState, useEffect, useContext} from 'react'

import queryString from 'query-string'

import io from 'socket.io-client'

import './Chat.css'

import InfoBar from "../InfoBar/InfoBar";

import Input from "../Input/Input";

import Messages from "../Messages/Messages";

let socket

const Chat = ({location}) => {

const [name, setName] = useState('')

const [room, setRoom] = useState('')

const [message, setMessage] = useState([])

const [messages, setMessages] = useState([])

const ENDPOINT = 'http://localhost:5001/'

useEffect(()=>{

const {name, room} = queryString.parse(location.search)

socket = io(ENDPOINT)

setName(name)

setRoom(room)

socket.emit('join', {name, room}, (error)=>{

if(error) alert(error)

})

return ()=>{

socket.emit('disconnect')

socket.off()

}

},[ENDPOINT, location.search])

useEffect(()=>{

socket.on('message', (message)=>{

setMessages([...messages, message])

})

}, [messages])

const sendMessage = (event)=>{

event.preventDefault()

if(message)

socket.emit('sendMessage', message, ()=> setMessage(''))

}

return (

<div className="outerContainer">

<div className="container">

<InfoBar room={room}/>

<Messages messages={messages} name={name}/>

<Input message={message} setMessage={setMessage} sendMessage={sendMessage}/>

</div>

</div>

);

};

export default Chat;

import React from 'react'

import './InfoBar.css'

import closeIcon from '../../icons/closeIcon.png'

import onlineIcon from '../../icons/onlineIcon.png'

const InfoBar = ({room}) => {

return (

<div className="infoBar">

<div className="leftInnerContainer">

<img className="onlineIcon" src={onlineIcon} alt="online"/>

<h3>{room}</h3>

</div>

<div className="leftInnerContainer">

<a href='/visit'><img src={closeIcon} alt="close"/></a>

</div>

</div>

)

}

export default InfoBar

import React from 'react'

import './Input.css'

const Input = ({message, setMessage, sendMessage}) =>(

<form className="form">

<input

className="input"

type="text"

placeholder="Type a message"

value={message}

onChange={(event)=> setMessage(event.target.value)}

onKeyPress={event => event.key === 'Enter' ? sendMessage(event):null}

/>

<button className="sendButton" onClick={(event)=> sendMessage(event)}>Send</button>

</form>

)

export default Input

import React, {useState} from 'react';

import {Link} from 'react-router-dom'

import './Join.css'

const Join = () => {

const [name, setName] = useState('')

const [room, setRoom] = useState('')

return (

<div className="joinOuterContainer">

<div className="joinInnerContainer">

<h1 className="heading">Join</h1>

<div><input placeholder="Name" className="joinInput" type="text" onChange={(event)=>setName(event.target.value)}/></div>

<div><input placeholder="Room" className="joinInput mt-20" type="text" onChange={(event)=>setRoom(event.target.value)}/></div>

<Link onClick={event=>(!name || !room)? event.preventDefault(): null} to={`/chat?name=${name}&room=${room}`}>

<button className="button mt-20" type="submit">Sign in</button>

</Link>

</div>

</div>

);

};

export default Join;

import React from 'react'

import ScrollToBottom from 'react-scroll-to-bottom'

import './Messages.css'

import Message from "./Message/Message";

const Messages = ({messages, name}) =>(

<ScrollToBottom className="messages">

{messages.map((message, i)=> <div key={i}><Message message={message} name={name}/></div>)}

</ScrollToBottom>

)

export default Messages

import React, {useContext} from 'react';

import {Switch, Route, Redirect} from 'react-router-dom'

import {publicRoutes} from "../Routes";

import {Context} from "../index";

const AppRouter = () => {

const {user} = useContext(Context)

return (

<Switch>

{publicRoutes.map(({path, Component}) =>

<Route key = {path} path = {path} component={Component} exact/>

)}

</Switch>

);

};

export default AppRouter;

import React from 'react';

import {Card, Col} from "react-bootstrap";

import {useHistory} from "react-router-dom";

import {DOCTOR\_ROUTE} from "../utils/consts";

const DocItem = ({doctor}) => {

const history = useHistory()

return (

<Col md={3} className="mt-4" onClick={()=> history.push(DOCTOR\_ROUTE + '/' + doctor.id)}>

<Card style={{width: 250, cursor:"pointer"}} border={"dark"}>

<div className="d-flex justify-content-between align-items-center">

Fullname: {doctor.fullname}<br/>

Speciality: {doctor.speciality}<br/>

Experience: {doctor.experience}<br/>

</div>

</Card>

</Col>

);

};

export default DocItem;

import React, {useContext, useEffect} from 'react';

import {observer} from "mobx-react-lite";

import {Context} from "../index";

import {Card, Row} from "react-bootstrap";

import DocItem from "./DocItem";

import {getAll} from "../https/docApi";

const DoctorLis = observer(() => {

const {doctor} = useContext(Context)

useEffect(()=>{

getAll().then(data=> {doctor.setDoctor(data);})

}, [])

return (

<Row className="d-flex">

{doctor.\_doctors.map(type=>

<DocItem key={type.id} doctor={type}/>

)}

</Row>

)

})

export default DoctorLis;

import React, {useContext, useEffect, useState} from 'react';

import {Context} from "../index";

import {Button, Container, Nav, Navbar, NavLink} from "react-bootstrap";

import {observer} from "mobx-react-lite";

import {useHistory, useLocation} from "react-router-dom";

import {

CHAT\_ROUTE,

CONSULTATION\_ROUTE,

HOMEANALYSE\_ROUTE,

LOGIN\_ROUTE,

REGISTER\_ROUTE,

VISIT\_ROUTE

} from "../utils/consts";

import {destroy} from "../https/userAPI";

const NavBar = observer(() => {

const {user} = useContext(Context)

const history = useHistory()

const location = useLocation()

const isLogin = location.pathname === LOGIN\_ROUTE

const isChat = location.pathname === CHAT\_ROUTE

function getLogin(){

if(!isChat) history.push(LOGIN\_ROUTE)

else alert('Exit from chat')

}

function getRegister(){

if(!isChat) history.push(REGISTER\_ROUTE)

else alert('Exit from chat')

}

function getTickets(){

if(!isChat) history.push(VISIT\_ROUTE)

else alert('Exit from chat')

}

function getHomeAnalyse(){

if(!isChat) history.push(HOMEANALYSE\_ROUTE)

else alert('Exit from chat')

}

function getChat(){

if(!isChat) history.push(CONSULTATION\_ROUTE)

else alert('Exit from chat')

}

async function deleteUser(){

if(!isChat) {

try{

let response = await destroy(user.\_user.id)

logout()

}catch(e){

alert('Error: cant delete this user')

}

}

else alert('Exit from chat')

}

function logout(){

if(!isChat) {

user.setUser({})

user.setIsAuth(false)

history.push(LOGIN\_ROUTE)

}

else alert('Exit from chat')

}

return (

<Navbar bg="success" variant="light">

<NavLink className="mx-auto">

{ user.isAuth ?

<Nav>

<Button onClick={getTickets} variant={"outline-dark"}>Tickets</Button>

<Button onClick={getHomeAnalyse} variant={"outline-dark"} className="mx-1">Home analyses</Button>

<Button onClick={getChat} variant={"outline-dark"} className="mx-1">Consultation</Button>

<Button onClick={logout} variant={"outline-dark"} >Log out</Button>

<Button onClick={deleteUser} variant={"outline-danger"} className="mx-1">Delete account</Button>

</Nav>

:

<Nav>

{ isLogin?

<Button onClick={getRegister} variant={"outline-dark"} className="mx-1">Register</Button>

:

<Button onClick={getLogin} variant={"outline-dark"} className="mx-1">Login</Button>

}

<Button onClick={getTickets} variant={"outline-dark"}>Tickets</Button>

<Button onClick={getHomeAnalyse} variant={"outline-dark"} className="mx-1">Home analyses</Button>

<Button onClick={getChat} variant={"outline-dark"}>Consultation</Button>

</Nav>

}

</NavLink>

</Navbar>

);

});

export default NavBar;

import React, {useContext} from 'react';

import {observer} from "mobx-react-lite";

import {Context} from "../index";

import {ListGroup} from "react-bootstrap";

const SpecBar = observer(() => {

const {doctor} = useContext(Context)

return (

<div>

<ListGroup>

{doctor.\_doctors.map(type=>

<ListGroup.Item

style={{cursor: 'pointer'}}

active={type.id === doctor.getSelectDoc().id}

onClick={()=> doctor.setSelectDoc(type)}

key={type.id}

>

{type.speciality}

</ListGroup.Item>

)}

</ListGroup>

</div>

)

})

export default SpecBar;

import React from 'react';

import {Card, Col} from 'react-bootstrap'

import {useContext} from "react";

import {Context} from "../index";

import {add} from "../https/visitApi";

import {useHistory} from "react-router-dom";

import {LOGIN\_ROUTE} from "../utils/consts";

import {observer} from "mobx-react-lite";

const VisitItem = observer(({visit}) => {

const {user} = useContext(Context)

const history = useHistory()

const addVisit = async()=>{

try{

let response = await add(visit.day, visit.begin, visit.end, user.\_user.id, visit.doctorId)

}catch(e){

alert(e.response.data.message)

history.push(LOGIN\_ROUTE)

}

}

return (

<Col md={4} className="mt-4" onClick={addVisit}>

<Card style={{cursor:"pointer"}}>

<div className="d-flex justify-content-between align-items-center">

Day: {visit.date}<br/>

Begin: {visit.begin}<br/>

End: {visit.end}

</div>

</Card>

</Col>

);

});

export default VisitItem;

import {$host} from "./index";

export const getAll = async()=>{

const {data} = await $host.get('/api/doctor/')

return data

}

import {$host} from './index'

export const setAnalyse = async(id, pulse, temperature, blood\_press, date)=>{

if(temperature === '') temperature = .0

if(pulse === '') pulse = 0

const {data} = await $host.post('/api/hAnalyse', {id, pulse, temperature, blood\_press, date})

return data

}

import axios from 'axios'

import jsonRoute from '../routes.json'

const $host = axios.create({

baseURL: jsonRoute.REACT\_APP\_API\_URL

})

const $authHost = axios.create({

baseURL:jsonRoute.REACT\_APP\_API\_URL

})

const authInterceptor = config=>{

config.headers.authorization = `Bearer ${localStorage.getItem('accessToken')}`

return config

}

$authHost.interceptors.request.use(authInterceptor)

export {$host, $authHost}

import {$authHost, $host} from './index'

import jwtDecode from "jwt-decode";

export const registration = async (fullname, birthdate, gender, telenumber, email, login, password, rep\_pass)=>{

const {data} = await $host.post('/api/user/register', {fullname, birthdate, gender, telenumber, email, login, password, rep\_pass})

}

export const login = async (login, password)=>{

const {data} = await $host.post('/api/user/login', {login, password})

localStorage.setItem('accessToken', data.accessToken)

return jwtDecode(data.accessToken)

}

export const destroy = async(id) => {

const {data} = await $host.delete(`/api/user/deleteUser/:${id}`)

localStorage.setItem('accessToken', '')

return data

}

import {$authHost, $host} from "./index"

export const add = async (date, begin, end, id, idDoc) =>{

const {data} = await $host.post('/api/visit/', {date: date, begin: begin, end: end, id: id, idDoc: idDoc})

}

export const getAll = async (id) =>{

const {data} = await $host.get(`/api/visit/:${id}`)

return data

}

import {makeAutoObservable} from "mobx";

export default class DoctorStore{

constructor() {

this.\_doctors = []

this.\_selectedDoc = {}

makeAutoObservable(this)

}

setDoctor(doctor){

this.\_doctors =doctor

}

getDoctor(){

return this.\_doctors

}

setSelectDoc(doctor){

this.\_selectedDoc = doctor

}

getSelectDoc(){

return this.\_selectedDoc

}

}

import {makeAutoObservable} from "mobx";

export default class UserStore{

constructor() {

this.\_isAuth = false

this.\_user = []

makeAutoObservable(this)

}

setIsAuth(bool){

this.\_isAuth = bool

}

setUser(user) {

this.\_user = user

}

get isAuth(){ return this.\_isAuth }

get user(){ return this.\_user }

}

import {makeAutoObservable} from "mobx";

import {getAll} from "../https/visitApi";

export default class VisitStore{

constructor(props) {

this.\_visites = []

this.\_selectedVisit = {}

makeAutoObservable(this)

}

setVisit(visit){

this.\_visites =visit

}

getVisit(){

return this.\_visites

}

setSelectVisit(visit){

this.\_selectedVisit = visit

}

getSelectVisit(){

return this.\_selectedVisit

}

}