Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования (СПП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Клиент-серверное приложение «****аренда участков»**

БГУИР КП 1-40 01 01 29 ПЗ

|  |  |  |
| --- | --- | --- |
|  |  | Студент: гр. 751001 А.А. Кимаев |
|  |  | Руководитель: Видничук В.Н. |

Минск 2019

СОДЕРЖАНИЕ

[Введение 5](#_Toc513665878)

[1 Анализ предметной области 6](#_Toc513665879)

[1.1 Обзор аналогов 6](#_Toc513665880)

[1.2 Постановка задачи 9](#_Toc513665881)

[2 Разработка программного средства 10](#_Toc513665882)

[2.1 Структура программы 10](#_Toc513665883)

[2.2 Интерфейс программного средства 12](#_Toc513665884)

[2.3 Логика приложения 15](#_Toc513665888)

[3 Тестирование программного средства](#_Toc513665890) 22

[4 Руководство пользователя 23](#_Toc513665891)

[Заключение 24](#_Toc513665894)

[Список использованных источников 25](#_Toc513665895)

[Приложение А. Исходный код программы 26](#_Toc513665896)

ВВЕДЕНИЕ

Технический прогресс в последние несколько десятков лет сильно изменил жизнь обычного человека. Цифровые технологии ускорили развитие различных областей науки, позволили повысить качество жизни. Немалое влияние развитие компьютеров оказало и на информационную сферу: стало куда легче искать и находить необходимую информацию, а после появления сети интернет стало возможным узнавать любую информацию из любой точки мира, появилась возможность узнавать новости из самых удалённых уголков планеты и быть в курсе последних событий на планете.

В том числе развитие информационной сферы крайне сильно повлияло и на торговую сферу: в современных рыночных условиях, когда высокий уровень конкуренции диктует внедрение более прогрессивных форм организации ведения бизнеса, когда устойчивый характер роста не является достоинством, а напротив — это единственный способ продолжать быть конкурентоспособным, особую актуальность и значимость приобретают именно информационные ресурсы, которые становятся надежной базой для эффективного управления деятельности предприятия.

Роль информационных технологий в работе современных предприятий постоянно возрастает. Наряду с традиционными направлениями их развития, связанными с автоматизацией процессов предприятий, для современных предприятий характерно применение новых высокоэффективных информационных технологий.

Правильно настроенная и интегрированная в деятельность компании информационная система может стать частью системы управления, реализуя информационную поддержку управленческих процессов и обеспечивая взаимодействие всех элементов организации для достижения её целей. Задача информационных технологий — разработать программу, в пределах которой будег анализироваться эффективность использования существующих систем, выявлять нереализованные выгоды и осуществлять меры по их реализации.

Это же касается и частных лиц, занимающихся торговлей, а также и простых граждан, которые тем или иным образом вовлечены в торговую сферу. Для того, чтобы минимизировать расход времени на поиск информации, нужны приложения, позволяющие оперативно получить доступ к необходимым источникам а также позволяющие получать информацию в реальном времени.

В настоящее время достаточно сложно оперативно получать полную информацию об арендуемых участках.

Целью данного проекта является разработка приложения, позволяющего изучать местность на предмет арендуемых участков, получать информацию о них и их владельцах.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На данный момент существует множество веб-приложений, позволяющих искать, арендовать и выставлять на продажу как земельные, так и жилые участки, за которыми закреплены кадастровые номера. Согласно принятому определению, кадастровый номер - это уникальное сочетание чисел, представляющее собой идентификационный номер определенного земельного надела на территории какой-либо страны. Рассмотрим несколько подобных веб-приожений.

* 1. Обзор аналогов

Веб-приложение torgi.gov.ru – российский сайт, для которого собрана гигантская база всевозможных участков со всей России. Для каждого участка имеется организатор торгов – лицо, непосредственно или опосредованно ведущее торги земельного участка, номер извещения, площадь участка, краткое описание границ земельного участка, его местоположение, а также предмет торга, такой как, например, цена земельного участка или стоимость аренды, ежегодный платёж или ежегодная арендная плата и начальная цена, с которой начинается торг. Изображение сайта приведено на рисунке ниже.

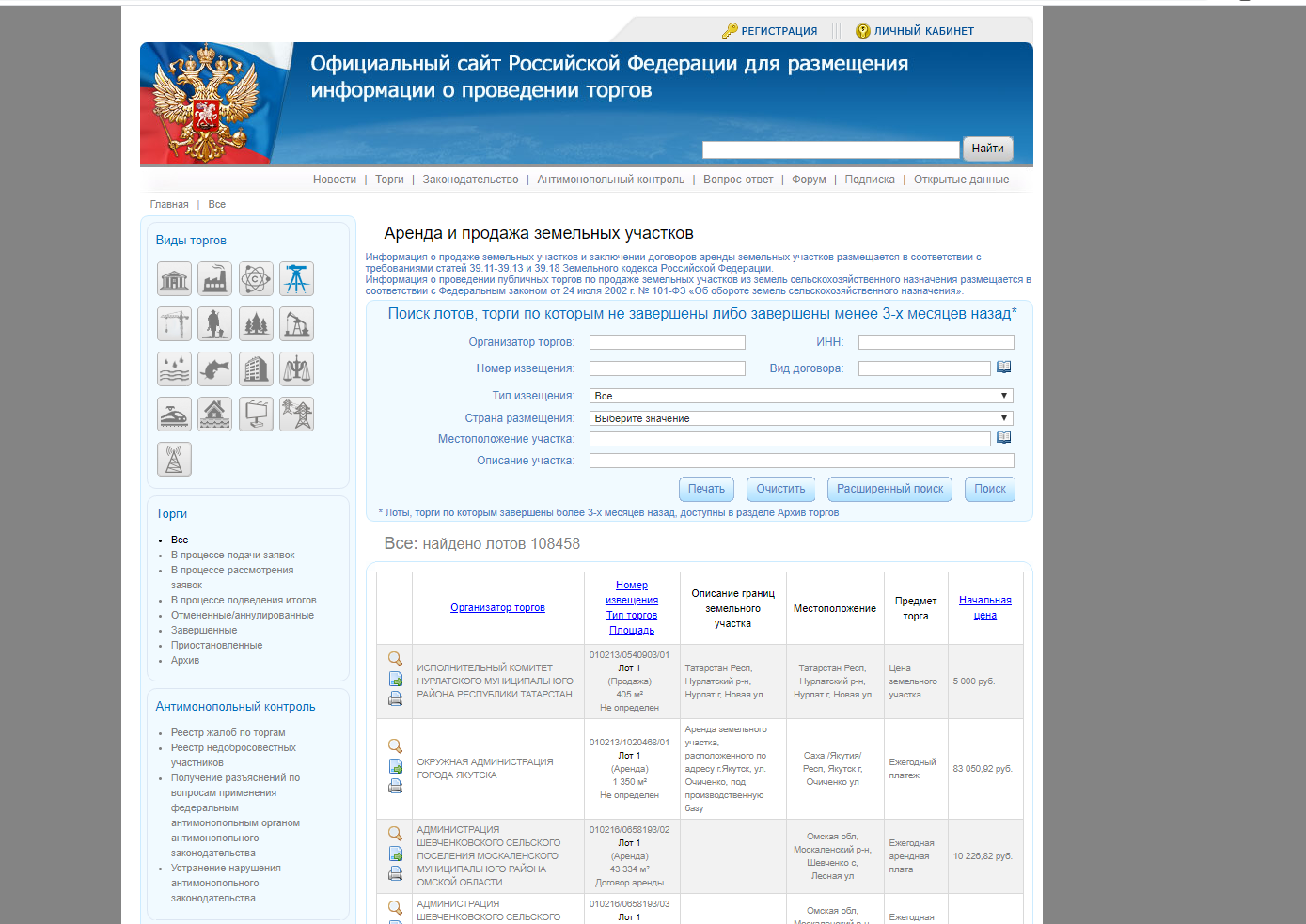


Рисунок 1.1 – сайт torgi.gov.ru

На данном сайте можно искать замельные участки по организатору торгов, ИНН – идентификационный номер плательщика – цифровой код, упорядочивающий учёт налогоплательщиков в Российской Федерации, по номеру извещения, стране размещения и многим другим параметрам. На этом же сайте присутствуют и различные другие виды товаров, помимо земельных участков, например, пользование участками недр, что даёт покупателю возможность добычи различных полезных ископаемых, в том числе газа и нефти.

Основным недостатком данного портала является устаревший интерфейс, а также большое количество всевозможнх вкладок, в которых легко запутаться.

Веб-приложение vl.nca.by – сайт, на котором возможно легко узнать так называемую кадастровую стоимость земель, а также налоговую базу. Кадастровая стоимость – это рыночная стоимость объекта недвижимости, которая устанавливается путем проведения государственной кадастровой оценки. Таким образом данный портал позволяет сравнительно легко получать информацию о стоисости различных земельных участков на территории Беларуси. Общий втд сайта вредставлен на рисунке 1.2.

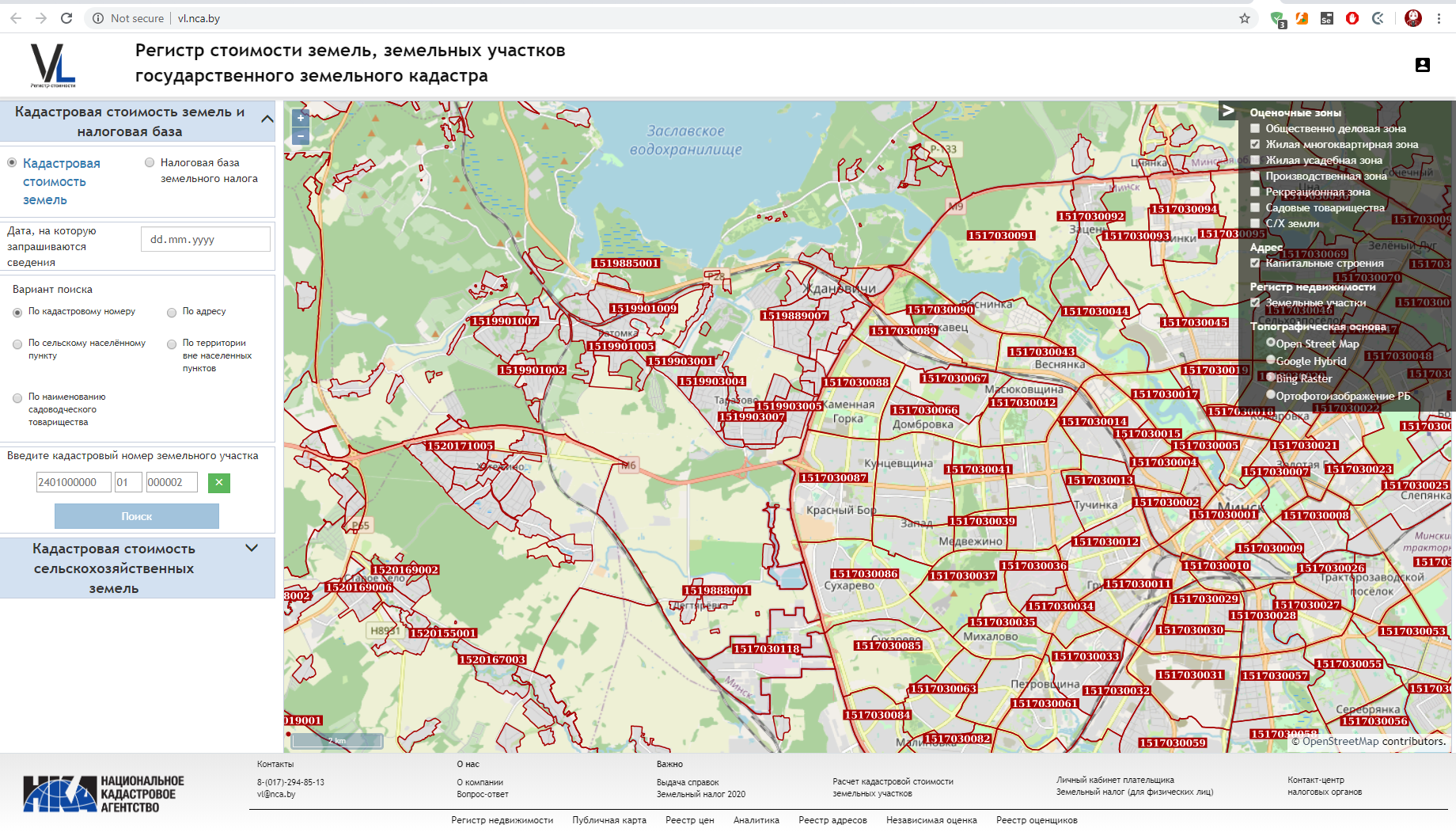


Рисунок 1.2 – сайт vl.nca.by

На данном портале можно получить информацию не только о земельных участках, но также и о участках из общественно-деловой зоны, о жилых, производственных и участках рекреационной зоны.

Данное веб-приложение также позволяет проводить поиск по кадастровому номеру, адресу и названию населённого пункта.

Интернет-портал [www.arendator.ru](http://www.arendator.ru) предоставляет возможности по аренде участков различных видов по Москве и Московской области, имеется достаточно широкий ассортимент различных объектов аренды, что продемонстрировано на рисунке 1.3.

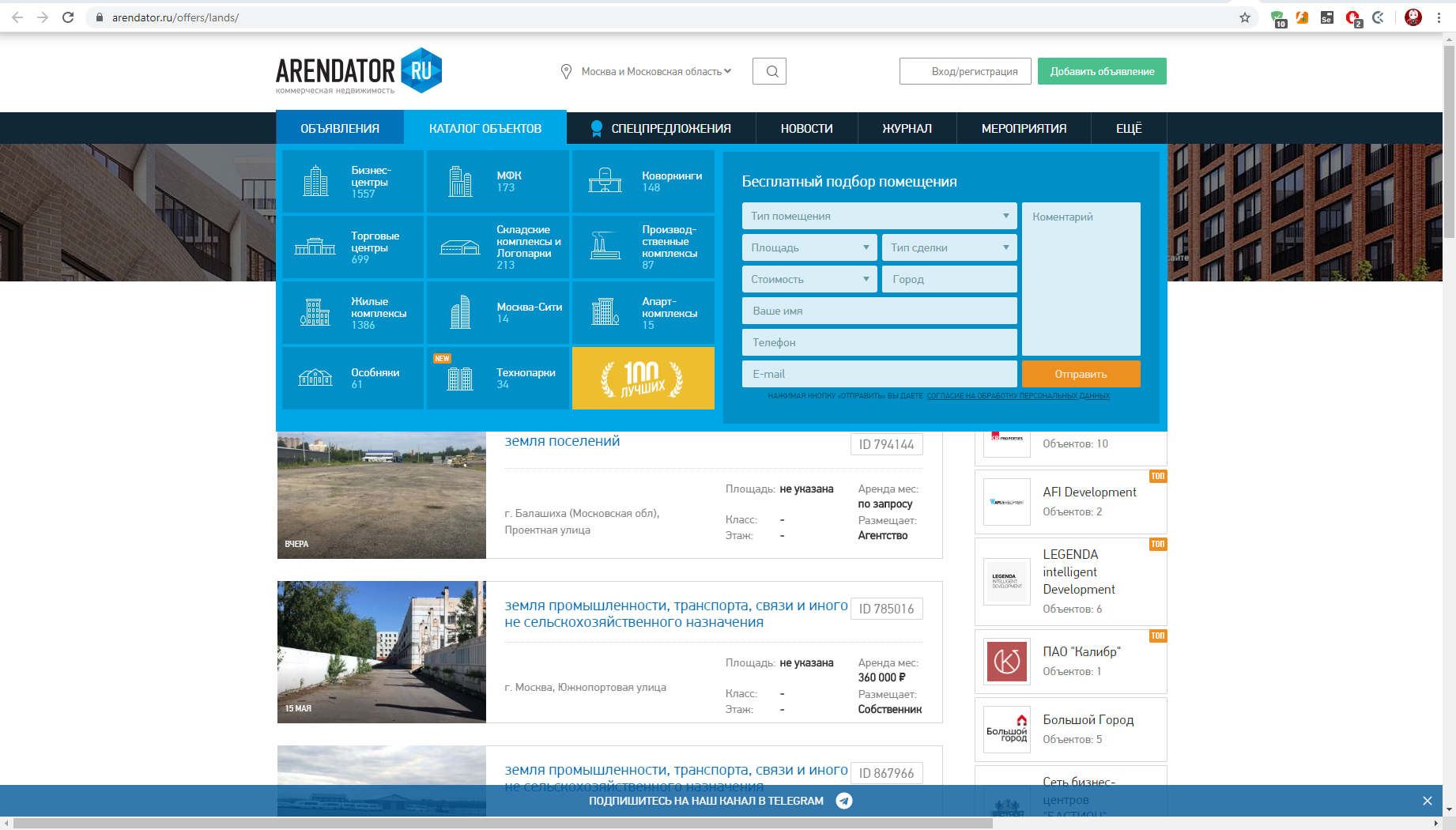


Рисунок 1.3 – сайт [www.arendator.ru](http://www.arendator.ru)

На этом сайте можно искать участки по различным критериям, таким как тип продажи: аренда или продажа, можно искать участки по площади, расположению и стоимости, а также назначению земли.

Имеется возможность получить дополнительную информацию о выбранном участке, напрмер подложка поверхности участка или наличие асфальтовой крошки. Подробное описание одного из арендуемых участков данного портала приведено на рисунке 1.4.

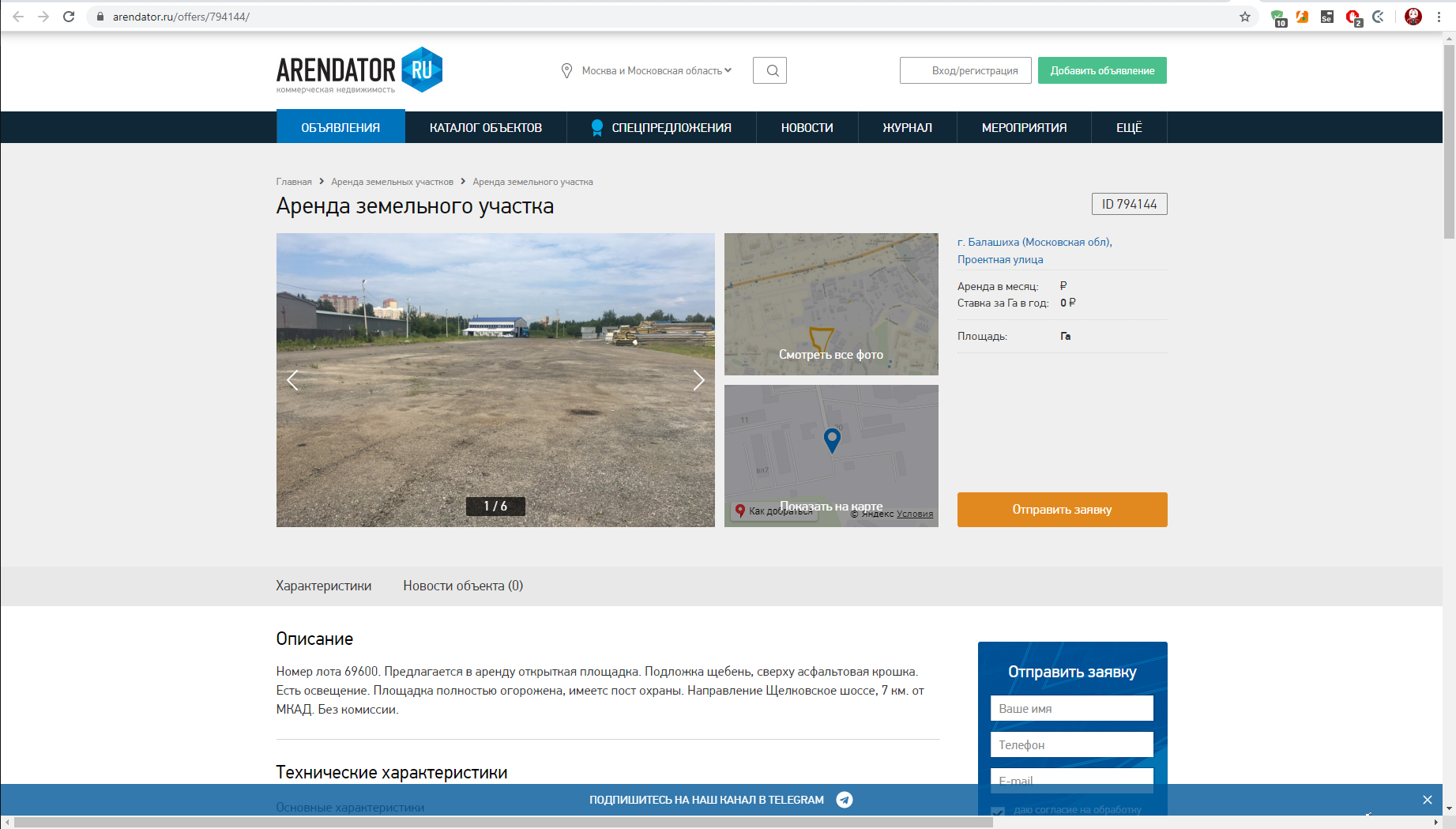


Рисунок 1.4 – один из участков на сайте [www.arendator.ru](http://www.arendator.ru)

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Plot rent», адаптированного под запуск в современных браузерах, в частности, на версиях Google Chrome 76.0.116.52 и старше

В процессе реализации будет разработана клиентская часть, а именно программный интерфейс и необходимая структура модулей для его реализации, а также серверная часть приложения, содержащая в себе слой общения с клиентской частью, слой бизнес-логики и слой DAO – data access object, или, слой общения с базой данных.

Целью данной курсовой работы является создание клиент-серверного приложения для быстрого поиска арендуемых участков.

**Задачи:**

* изучить способы построения карт и систем добавления на них пользовательских элементов для отображения арендуемых участков;
* реализовать интерфейс веб-приложения интуитивно понятным и простым;
* реализовать функцию просмотра информации об участке и его владельце;
* реализовать возможность добавления документов, подтверждающих право пользователя распоряжаяться участком

**Необходимые ресурсы**

Для разработки приложения использован язык JavaScript. В разработке клиентской части приложения использован фреймворк ReactJS, а серверной – ExpressJS. Редактор исходного кода – Webstorm. Выбор языка связан с популярностью и широким спектром возможностей в разработке веб-ориентированных приложений, как и в области интерфейсов, так и в бизнес-логике приложения.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

Структура программы будет представлена двумя частями: клиентской и серверной.

**Архитектура сервера**

Для осуществления функций регистрации, авторизации, аутентификации, редактирования профиля, сохранения сообщений чатов, отслеживания местоположения пользователей, работы с глобальными настройками пользователя был реализован сервер приложения, основанный на архитектуре REST и реализованный при помощи фреймфорка вебприложений ExpressJS для платформы Node.js. Данная архитектура была выбрана для повышения производительности сервера и упрощения его структуры. REST архитектура подразумевает отсутствие состояния, это означает, что сервер не хранит у себя никаких данных о клиенте, а каждый новый запрос содержит в себе достаточно информации для его выполнения. Взаимодействие между клиентом и сервером происходит по протоколу HHTP, используя методы данного протокола, для создания, изменения и удаления объектов. Объектами в данном контексте выступают сообщения пользователей, чаты, настройки поиска и сами объекты пользователей. Они хранятся в базе данных, которая находится под управлением реляционной СУБД MySQL. Структура базы данных приведена на рисунке ниже.

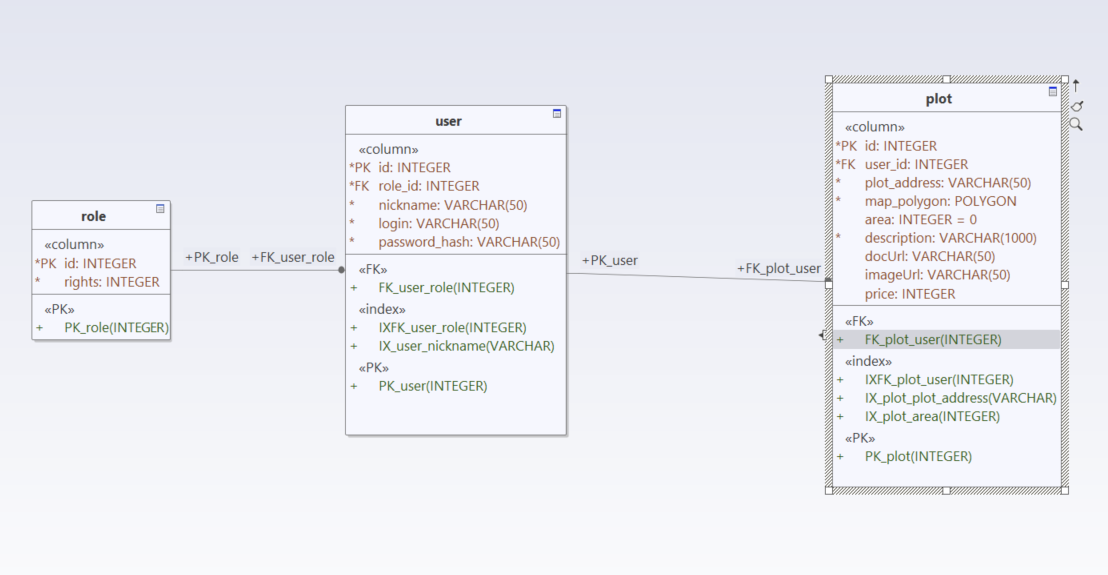
****

Рисунок 2.1 – Архитектура базы данных.

Использование фреймворка ExpressJS подразумевает использование так называемого middleware. Middleware – функции, которые последовательно обрабатывают объекты запроса и ответа, модифицируя их в соответствии со своим назначением. В виде middleware, например, на сервере реализована авторизация, подразумевающая проверку специального объекта-токена при попытке доступа к по адресу, который не относится к общедоступным. При проверке происходит декодирование токена, который был ранее выдан пользователю при процессе входа и прикреплен к заголовку запроса “x-auth”, после чего из декодированного токена извлекается email пользователя, обратившегося к серверу, и определяется, может ли пользователь получить доступ к запрашиваемому ресурсу.

Для взаимодействия с объектами базы данных, которое осуществляется в объектах-роутерах сервера, используются объекты DAO. Класс GenericDao описывает базовые операции со всеми объектами, такие как получение всех объектов определенного типа, получение отдельного объекта по его идентификатору, создание объекта, обновление объекта. Наличие в каждой сущности полей, предоставляющих базовые запросы для работы с этими объектами в базе данных, позволяют реализовать класс GenericDao таким образом, чтобы снизить количество кода, необходимое на реализацию конкретного DAO для заданного типа объекта, заменяя все повторяющиеся методы для конкретных классов обычным наследованием.

Помимо GenericDao присутствуют ещё 2 DAO:

* UserDao – отвечает за работу с объектами пользователя, реализует в каждом методе проверки, необходимые для выполнения того или иного метода GenericDao.
* PlotDao – отвечает за работу с объектами участка, реализует 2 дополнительных метода обновления участка: при обновлении фотографии и обновлении документа, подтверждающего право на владение.

Как упоминалось выше, для обработки пользовательских запросов по определенным URL’ам используются объекты фреймворка express-роутеры. Они обрабатывают HHTP запросы, такие как POST, GET, PUT, которые выполняет клиент для создания, обновления и получения объектов, необходимых ему локально.

**Архитектурxа клиента**

Клиент представлен в виде Single Page Application (SPA), что подразумевает собой реализацию веб-приложения, исполняемого браузером, как одну страницу. С точки зрения пользователя, данная технология привлекает в первую очередь быстротой отклика на действия в пользовательском интерфейсе, так как не требуется полная или даже частичная перезагрузка веб-страницы с сервера, а все визуальные элементы конструируются прямо в браузере с помощью JavaScript путем манипуляций с DOM-структурой документа. Таким образом, веб-приложение становится очень похожим на обычное приложение, устанавливаемое на персональный компьютер, при этом сохраняя браузер как исполнителя, что позволяет пользоваться приложением с любого устройства, на котором может быть установлен браузер.

Поскольку для разработки клиента был выбран фреймворк ReactJS, клиент разбит на отдельные React-компоненты, которые содержат в себе методы рендера и обработчики, которые вызываются в зависимости от действий пользователя. Разбиение на компоненты позволяет разделить ответственность отдельных частей программы и потенциально использовать их повторно в различных местах. Точка входа в приложение – компонент App. Поскольку данный компонент включает в себя компоненты авторизации, регистрации, и основной компонент – MainApp, в его состоянии хранится текущий пользователь, который устанавливается при успешной авторизации через компонент авторизации. Компонент авторизации обращается к серверной части приложения, передавая данные авторизации, которые проходят проверку, после чего, в случае успешной авторизации, клиенту выдаётся JSON Web Token, который отправляется в заголовках всех запросов на сервер для подтверждения личности пользователя.

Компонент MainApp, в свою очередь, использует следующие компоненты:

* Map – главный комнонент программы, содержит карту с выделенными на ней продаваемыми участками различных пользователей.
* Topbar – шапка страницы, содержит логотип и кнопку открытия боковой панели
* Sidepane – боковая панель, содержит кнопки открытия модального окна профиля и выхода из аккаунта.
* Модальные окна – несколько компонентов, отвечающих за вывод на экран и редактирование различной информации об объектах программы: пользователях, участках и документах, подтверждающих право на владение.
  1. Интерфейс программного средства

При первом попадании на страницу веб-приложения открывается экран авторизации, где также можно зарегистрироваться.

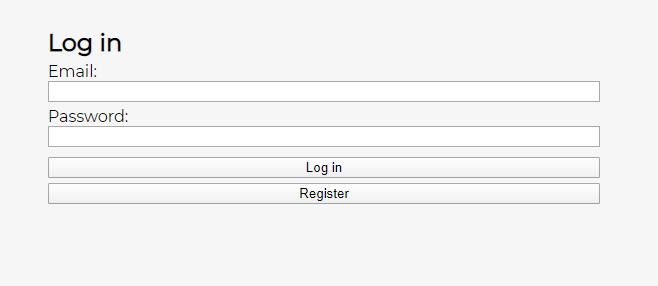


Рисунок 2.2 – Экран авторизации.

После нажатия на кнопку Register появляется экран регистрации, на котором требуется заполнить некоторые поля.

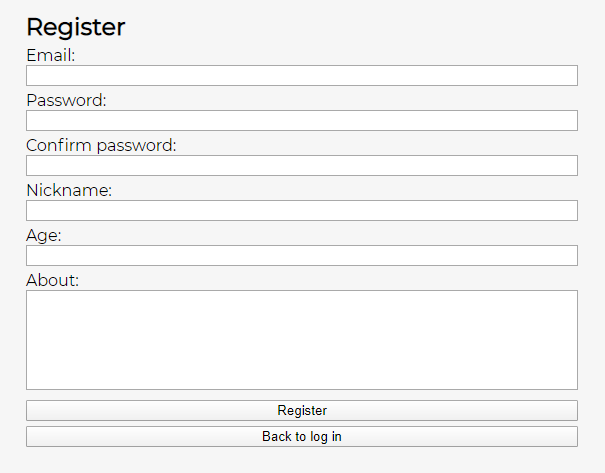


Рисунок 2.3 – Экран регистрации

После успешной авторизации либо регистрации открывается экран с картой, на которой синим цветом отображены участки, принадлежащие текущему пользователю, а красным – участки, принадлежащие всем остальным.

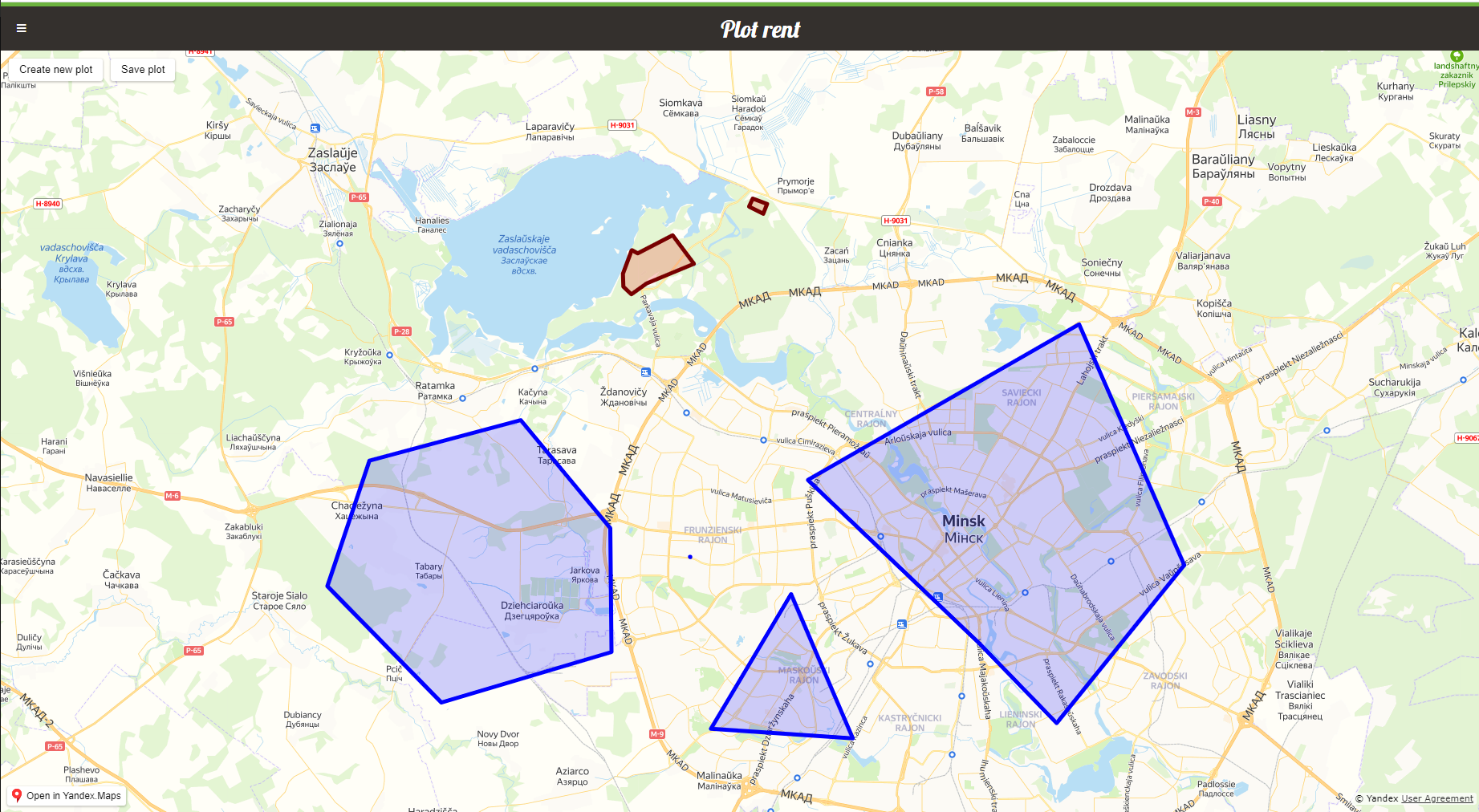


Рисунок 2.4 – Главный экран с картой и участками

После нажатия на любой участок, появляется небольшое окно с информацией об участке. При этом, если участок принадлежит текущему пользователю в окне имеется дополнительная кнопка Edit, после нажатия на которую появляется модльное окно, в котором пользователь может поменять значения атрибутов участка или удалить его.

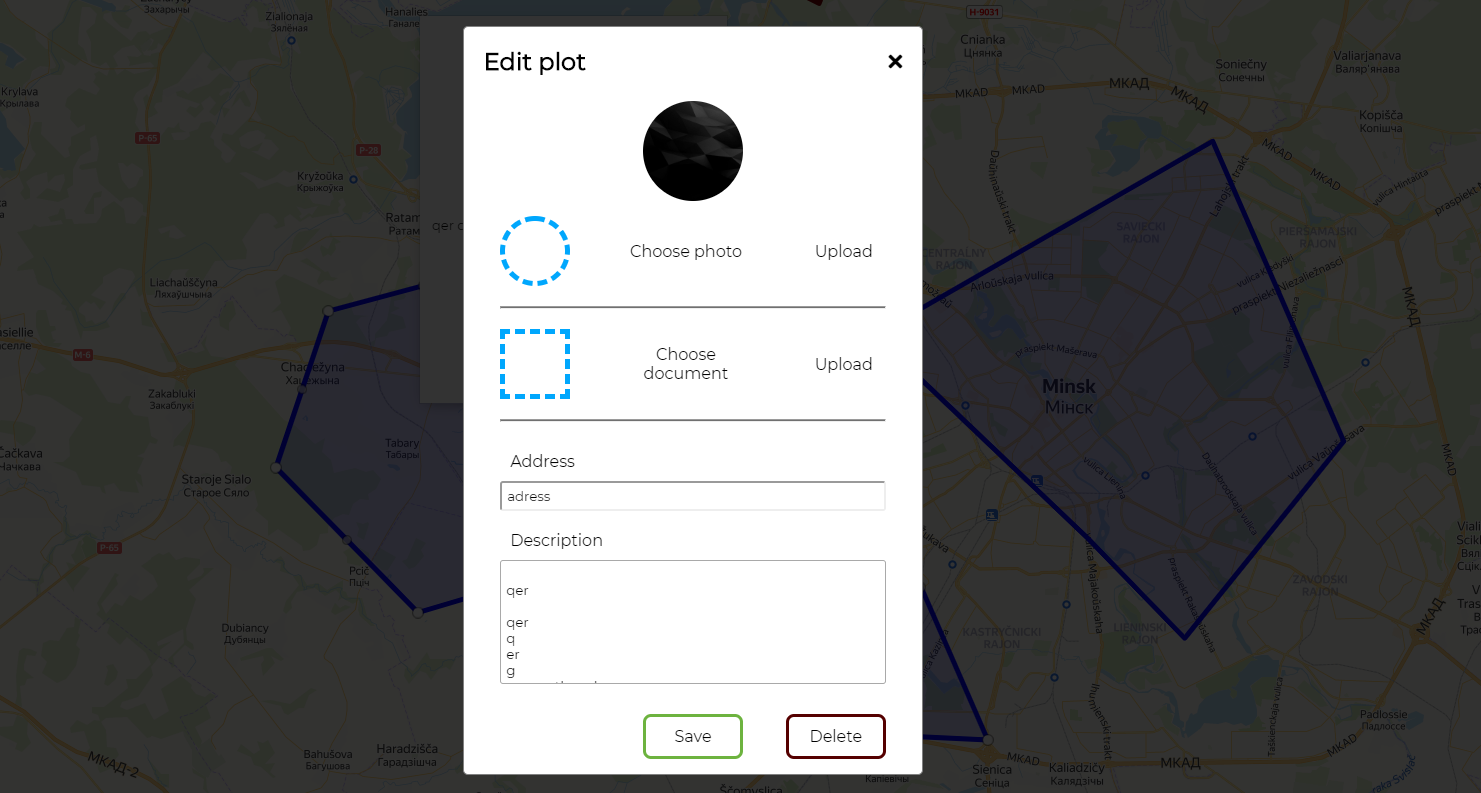


Рисунок 2.5 – Модальное окно редактирования участка

Помимо кнопки Edit информационное окно участка содержит также кнопку Plot Details, после нажатия на которую открывается модальное окно с .pdf документом, который был прикреплён пользователем к участку, как документ подтверждающий право на владение. Если такого документа нет, будет отображена надпись «No document available».

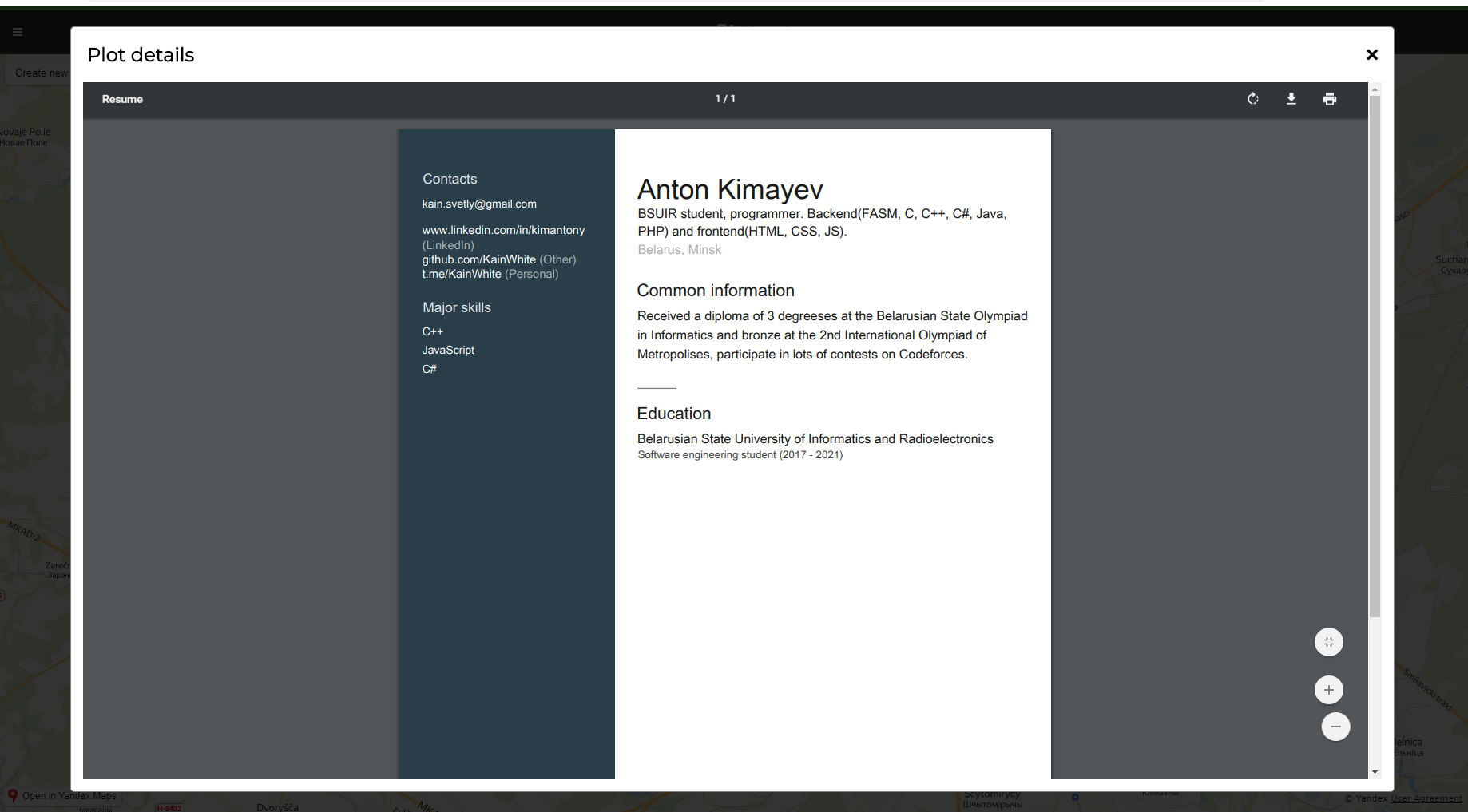


Рисунок 2.6 – Модальное окно документа владения участком

В верхнем левом углу экрана имеются две кнопки: «Create new plot» и «Save plot», они нужны соответственно для создания нового участка и его сохранения. Открыв боковую панель можно отредактировать свой профиль, нажав на кнопку Profile и выйти из приложения, нажав Log out.

* 1. Логика приложения

Точка входа в приложение на стороне клиента – файл index.js:

import React from 'react';

import ReactDOM from 'react-dom';

import './index.scss';

import {App} from './app/app.js';

ReactDOM.render(<App />, document.getElementById('root'));

В нём происходит рендер компонента App внутри html элемента с id = root. App, в свою очередь, в зависимости от того, авторизован ли пользователь приложения или нет, рендерит или экран авторизации, либо MainApp, который является основным компонентом программы.

MainApp далее рендерит компоненты Topbar, Sidepane и YandexMap, а также, в нём происходит отображение модальных окон пользователей : EditableUserProfile и UserProfile. Ниже приведена часть кода методов MainApp.render(), выполняющего рендер компонента и renderSwitchModalWindow, в котором решается, какое модальное окно будет отображено.

renderSwitchModalWindow(modalWindow) {

switch (modalWindow) {

case ModalWindowEnum.none:

return;

case ModalWindowEnum.profileSelf:

return <EditableUserProfile … />;

case ModalWindowEnum.profileUser:

return <UserProfile … />;

default:

console.error("Unsupported modal window " + modalWindow);

}

}

render() {

return (

<React.Fragment>

<Sidepane sidepaneStyle={this.state.sidepaneStyle} … />

<div className={"main"} style={this.state.mainStyle}>

<Topbar toggleSidepane={() => this.toggleSidepane()}/>

<YandexMap … />

</div>

{ this.renderSwitchModalWindow(this.state.modalWindowEnum) }

</React.Fragment>

);

}

Компонент YandexMap отвечает за работу с api Яндекс-карт, последние не поддерживают React-компоненты, поэтому внутренние окна для описания участков необходимо писать в процедурном javascript стиле. Ниже приведена процедура, инициализирующая разметку всплывающего окна balloon и возвращающая многоугольник:

function PlotPolygon(ymaps, props) {

const makeLayout = (layoutFactory) => {

const plotBalloon = new PlotBalloon(props);

const Layout = layoutFactory.createClass(

plotBalloon.renderAsString(),

{

build: function() {

Layout.superclass.build.call(this);

… // здесь происходит выставление обработчиков кнопок

},

clear: function() {

… // здесь происходит снятие обработчиков кнопок

Layout.superclass.clear.call(this);

},

user: props.user,

plot: props.plot,

profileClick: props.profileClick,

editPlotClick: props.editPlotClick,

showPlotDetailsClick: props.showPlotDetailsClick,

});

return Layout;

};

return new ymaps.Polygon(

props.geometry,

props.properties,

{

...props.options,

balloonContentLayout: makeLayout(ymaps.templateLayoutFactory),

balloonPanelMaxMapArea: 0,

},

);

}

Взаимодействие с сервером происходит с помощью библиотеки axios, вызовы которой происходят в обработчиках кликов в модальных окнах, отвечающих за редактирование сущностей, например, EditablePlot и в обработчиках кнопок, которые отвечают за добавление и сохранение участков. Ниже приведён код одного из вызовов серверного api, в котором происходит обновление участка.

try {

const response = await API.put(

`plots/${plotModel.id}`,

plotModel,

{headers: {"Content-Type": "application/json"}}

);

if (!response.error) {

return response.data.plot;

} else {

console.error(response.error);

}

} catch (e) {

console.error(e);

}

На стороне сервера точка входа в приложение – файл app.js, в котором устанавливается соединение с базой данных, настраиваются политики доступа cors, назначаются другие middleware, в том числе и роутеры, которые и обрабатывают запросы с клиента. Ниже приведён код plotRouter’a, ответственного за обработку запросов, связанных с участками.

const PlotDAO = require('../dao/plot-dao');

const PlotModel = require('../dto/plot-model');

const express = require('express');

const bodyparser = require('body-parser');

const plotRouter = express.Router();

plotRouter.use(bodyparser.json());

//! base route '/plots'

/\*\*

\* Get all plots

\*/

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

let plots = await PlotDAO.getAll();

if (!plots.error) {

plots = plots.map(plot => new PlotModel(plot));

}

handleDefaultDaoResponse(plots, res, 'plots');

});

/\*\*

\* Get plot by id

\*/

plotRouter.get('/:plotId', async function(req, res, next) {

let plot = await PlotDAO.getById(req.params.plotId);

if (!plot.error) {

plot = new PlotModel(plot);

}

handleDefaultDaoResponse(plot, res, 'plot');

});

/\*\*

\* Create plot

\*/

plotRouter.post('/create', async function(req, res, next) {

let plot = await PlotDAO.create(req.body);

if (!plot.error) {

plot = new PlotModel(plot);

}

handleDefaultDaoResponse(plot, res, 'plot');

});

/\*\*

\* Update plot

\*/

plotRouter.put('/:plotId', async function(req, res, next) {

let plot = await PlotDAO.update(req.body);

if (!plot.error) {

plot = new PlotModel(plot);

}

handleDefaultDaoResponse(plot, res, 'plot');

});

function handleDefaultDaoResponse(daoResponse, res, propertyName) {

if (daoResponse.error) {

res.json(daoResponse);

} else {

res.json({[propertyName]: daoResponse});

}

}

Каждый роутер объявляет функции-обработчики для приходящих с клиента http запросов. В самих функциях-обработчиках вызываются соответствующие DAO, которые наследуются от GenericDao, код которого представлен ниже.

const mysql = require('mysql');

const util = require('util');

const config = require('../app-config.json');

class GenericDao {

static connection;

static establishConnection() {

const connection = mysql.createConnection(config.DB);

this.connection = {

connect: util.promisify(connection.connect).bind(connection),

query: util.promisify(connection.query).bind(connection),

close: util.promisify(connection.end).bind(connection),

escape: connection.escape.bind(connection),

};

this.connection.connect();

}

static async getAll() {

const sql = `SELECT \* FROM ${this.entityClass.tableName}`;

const rows = await this.connection.query(sql);

return rows.map(row => new this.entityClass(row));

}

static async getById(id) {

const sql = `SELECT \* FROM ${this.entityClass.tableName} WHERE id = ?`;

const rows = await this.connection.query(sql, [id]);

return new this.entityClass(rows[0]);

}

static async create(entityData) {

const sql = this.entityClass.getCreateSql();

const row = await this.connection.query(sql,

this.entityClass.getCreatePlaceholdersArray(entityData));

return await this.getById(row.insertId);

}

static async update(entity) {

const oldEntity = await this.getById(entity.id);

if (!oldEntity) {

return {error: `Entity ${this.entityClass.className} with

id = ${entity.id} doesn't exist!`};

}

const updateEntity = {

...oldEntity,

...entity,

};

const sql = this.entityClass.getUpdateSql();

await this.connection.query(sql,

this.entityClass.getUpdatePlaceholdersArray(updateEntity));

return await this.getById(entity.id);

}

static async delete(id) {

return await this.connection.query(

`DELETE FROM ${this.entityClass.tableName} WHERE id=?`, [id]);

}

}

module.exports = GenericDao;

GenericDao объявляет CRUD методы, которые используются потомками, у которых объявлено поле entityClass, значение которого соответствует сущности из базы данных. Таким образом у PlotDao entityClass является Plot, которые реализует методы get\*\*\*Sql и get\*\*\*Placeholders, которые применяются в GenericDao для получения цельного запроса и отправки его в базу данных.

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

При разработке пользовательского интерфейса клиентской части приложения был замечен дефект в поведении кнопки «Create new plot»: при нескольких последовательных нажатиях создавались новые объекты участков в базе данных. На рисунке ниже показан данный недочёт.

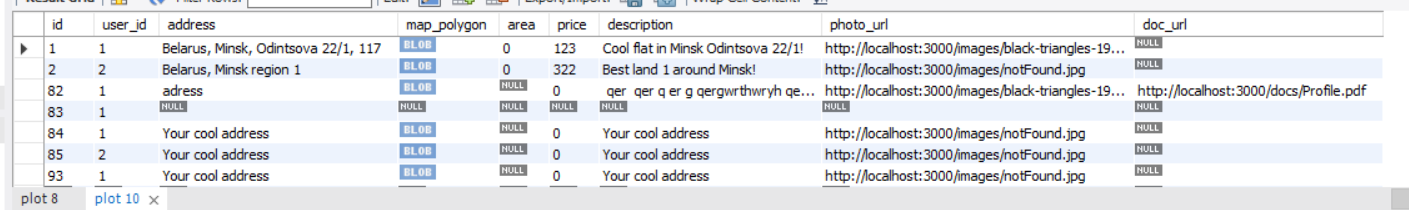


Рисунок 3.1 – Лишние созданные участки в базе данных

Данный дефект был устранён путём добавления в состояние компонента карты поля редактируемого участка и проверок на то, чтобы к моменту повторного нажатия на кнопку «Create new plot» у многоугольника участка было бы как минимум 3 вершины.

При тестировании удаления и редактирования участков проблем выявлено не было.

Остальные проблемы были связаны с недочётами на стадии проектирования и были устранены на стадии разработки.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для успешной работы необходимо иметь подключение к сети Интернет и веб браузер.

Чтобы использовать приложение необходимо пройти регистрацию, после чего пользователь будет автоматически залогинен с помощью созданного аккаунта.

Использование самого проиложения достаточно простое: для просмотра информации об интересующем участке необходимо кликнуть по его многоугольнику; для того чтобы узнать сведения о владельце, необходимо в открывшемся окне нажать на кнопку «Show profile»; чтобы удостовериться в принадлежности участка пользователю необходимо нажать на кнопку «Show details».

Для выставления своего участка в аренду или на продажу необходимо нажать на кнопку в верхнем левом углу «Create plot» и далее одиночными нажатиями левой кнопки мыши выставить все требуемые вершины многоугольника участка, после чего, либо нажать на кнопку «Save plot» в верхнем левом углу, либо нажать на любую вершину и в появившемся окне нажать на кнопку «Finish», после чего можно перейти к окну редактирования участка, кликнув по многоугольнику участка и в открывшемся окне нажав на кнопку «Edit».

Для редактирования уже созданных участков требуется кликнуть по многоугольнику участка и в открывшемся окне нажать на кнопку «Edit».

Для того, чтобы изменить свей профиль, требуется открыть боковую панель, нажав на кнопку в виде трёх чёрточек в верхнем левом углу и нажать на кнопку «Profile».

Чтобы выйти из текущего аккаунта нужно в боковой панели нажать на кнопку «Log out».

**ЗАКЛЮЧЕНИЕ**

В результате данной курсовой работы было разработано веб-приложение Plot rent, и выполнены следующие задачи:

* изучены способы построения карт и систем добавления на них пользовательских элементов для отображения арендуемых участков;
* интерфейс веб-приложения реализован интуитивно понятным и простым, для выполнения любого действия не требуется более нескольких нажатий по выделяющимся кнопкам;
* реализована функцию просмотра информации об участке и его владельце;
* реализована возможность добавления документов, подтверждающих право пользователя распоряжаяться участком;
* реализована как клиентская, так и серверная часть приложения, обе части имеют модульную, легко внедряемую и поддерживаемую архитектуру.

Приложение имеет большой потенциал для развития, впоследствии могут быть добавлены разные типы участков, способы фильтрации участков, методы нахождения лучших участков по соотношения цена/качество, чаты между пользователями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Форум для программистов. Электронные данные. Режим доступа: <https://stackoverflow.com> – Дата доступа: 17.03.2020.
2. Яндекс-api. Электронные данные. Режим доступа: <https://tech.yandex.com/> – Дата доступа: 20.03.2020
3. Интеграция Яндекс-api и React. Электронные данные. Режим доступа: https://react-yandex-maps.now.sh/ – Дата доступа: 10.04.2020.
4. ReactJS – документация к фреймворку. Электронные данные. Режим доступа: https://reactjs.org/docs/ – Дата доступа: 10.02.2020.

ПРИЛОЖЕНИЕ А

Исходный код программы

app.js - client

import React from 'react';

import {AuthorizedEnum} from './authorized-enum.js';

import {MainApp} from './main-app/main-app'

import {AuthScreen} from './login-and-register-screen/login-and-register-screen';

import API from './api';

import './app.scss';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

authorized: AuthorizedEnum.unauthorized,

};

}

onAuthorized = (newState, user, userSettings, token) => {

API.defaults.headers.common['x-auth'] = token;

this.setState({

authorized: newState,

currentUser: user,

userSettings: userSettings,

});

};

updateCurrentUser = (newUser) => {

this.setState({currentUser: newUser});

};

updateUserSettings = (newSettings) => {

this.setState({userSettings: newSettings});

};

render() {

return (

<React.Fragment>

{

this.state.authorized === AuthorizedEnum.unauthorized ?

<AuthScreen authHandler={this.onAuthorized}/> :

<MainApp currentUser={this.state.currentUser}

userSettings={this.state.userSettings}

updateCurrentUser={this.updateCurrentUser}

updateUserSettings={this.updateUserSettings}

logoutClick={() => {

this.setState({

authorized: AuthorizedEnum.unauthorized,

currentUser: undefined});

delete API.defaults.headers.common['x-auth'];

}}/>

}

</React.Fragment>

);

}

}

export {App};

main-app.js

import React from 'react';

import {MainContainerEnum} from './main-container-enum';

import {ModalWindowEnum} from './modal-window-enum';

import {Topbar} from './topbar/topbar';

import {Sidepane} from './sidepane/sidepane';

import {YandexMap} from './map/map';

import {ChatScreen} from "../chat/chat-screen";

import {EditableUserProfile} from './profile-screen/editable-user-profile';

import {UserProfile} from './profile-screen/user-profile';

import {SettingsScreen} from './settings-screen/settingsScreen';

import "./main-app.scss";

import {EditablePlot} from './map/plot-modals/editable-plot/editable-plot';

class MainApp extends React.Component {

constructor(props) {

super(props);

this.state = {

sidepaneVisible: false,

sidepaneStyle: {

width: 0,

},

mainStyle: {

marginLeft: 0,

},

mainContainer: MainContainerEnum.map,

modalWindowEnum: ModalWindowEnum.none,

}

}

toggleSidepane(close) {

const sidepaneStyle = {};

const mainStyle = {};

if(close) {

sidepaneStyle.width = '0';

mainStyle.marginLeft = '0';

this.setState({

sidepaneVisible: false,

sidepaneStyle: sidepaneStyle,

mainStyle: mainStyle,

});

return;

}

if (this.state.sidepaneVisible) {

sidepaneStyle.width = '0';

mainStyle.marginLeft = '0';

} else {

sidepaneStyle.width = '200px'; // sidepane.offsetWidth + 'px';

mainStyle.marginLeft = '200px';

}

this.setState((state) => {

return {

sidepaneVisible: !state.sidepaneVisible,

sidepaneStyle: sidepaneStyle,

mainStyle: mainStyle,

}

});

}

showChat(userToChat) {

this.setState({

mainContainer: MainContainerEnum.chat,

userToChat: userToChat,

});

this.toggleSidepane(true);

}

showMap() {

this.setState({

mainContainer: MainContainerEnum.map,

});

this.toggleSidepane(true);

}

renderSwitchModalWindow(modalWindow) {

switch (modalWindow) {

case ModalWindowEnum.none:

return;

case ModalWindowEnum.profileSelf:

return <EditableUserProfile

user={this.props.currentUser}

updateCurrentUser={this.props.updateCurrentUser}

onClose={() => this.setState({

modalWindowEnum: ModalWindowEnum.none,

})}

/>;

case ModalWindowEnum.profileUser:

return <UserProfile

user={this.state.userToShowProfile}

chatClick={(userToChat) => this.showChat(userToChat)}

onClose={() => this.setState({

modalWindowEnum: ModalWindowEnum.none,

})}

/>;

case ModalWindowEnum.settings:

return <SettingsScreen

currentUser={this.props.currentUser}

userSettings={this.props.userSettings}

onClose={() => this.setState({

modalWindowEnum: ModalWindowEnum.none,

})}

updateUserSettings={this.props.updateUserSettings}

/>;

default:

console.error("Unsupported modal window " + modalWindow);

}

}

render() {

return (

<React.Fragment>

<Sidepane sidepaneStyle={this.state.sidepaneStyle}

profileClick={() => this.setState({

modalWindowEnum: ModalWindowEnum.profileSelf,

})}

settingsClick={() => this.setState({

modalWindowEnum: ModalWindowEnum.settings,

})}

chatsClick={() => this.showChat(null)}

mapClick={() => this.showMap()}

logoutClick={this.props.logoutClick}/>

<div className={"main"} style={this.state.mainStyle}>

<Topbar toggleSidepane={() => this.toggleSidepane()}/>

{

this.state.mainContainer === MainContainerEnum.map ?

<YandexMap currentUser={this.props.currentUser}

userSettings={this.props.userSettings}

updateCurrentUser={this.props.updateCurrentUser}

profileClick={(userToShowProfile) => {

this.setState({

userToShowProfile: userToShowProfile,

modalWindowEnum: ModalWindowEnum.profileUser,

});

}}/> :

<ChatScreen currentUser={this.props.currentUser}

userToChat={this.state.userToChat}/>

}

</div>

{ this.renderSwitchModalWindow(this.state.modalWindowEnum) }

</React.Fragment>

);

}

}

export {MainApp}

map.js

import './map.scss';

import React from 'react';

import {YMaps, Map, Polygon, Button} from 'react-yandex-maps';

import {PlotPolygon} from './plot-polygon/plot-polygon';

import {EditablePlot} from './plot-modals/editable-plot/editable-plot';

import {PlotDocument} from './plot-modals/plot-document/plot-document';

import {PlotService} from '../plot-service/plot-service';

import {PlotModel} from '../../common/dto/plot-model';

import API from '../../api';

class YandexMap extends React.Component {

constructor(props) {

super(props);

this.state = {

currentUserPlots: [],

};

this.userPlots = [];

this.users = [];

}

setCurrentPosition = () => {

navigator.geolocation.getCurrentPosition((position) => {

this.setState({

userPosition: [

position.coords.latitude, position.coords.longitude

],

});

});

};

setPlots = async () => {

const plots = await PlotService.getAll();

this.userPlots = plots.filter(plot =>

plot.userId !== this.props.currentUser.id);

this.setState(() => {

const currentUserPlots = [];

plots.filter(plot =>

plot.userId === this.props.currentUser.id).forEach(plot => {

currentUserPlots[plot.id] = plot;

});

return {

currentUserPlots: currentUserPlots,

}

});

};

setUsers = async () => {

const users = (await API.get(

`users/`,

{headers: {"Content-Type": "application/json"}}

)).data.users;

users.forEach((user) => this.users[user.id] = user);

};

newPlotClick = async () => {

if (!this.state.ymaps) {

console.error('no ymaps set');

return;

}

if (this.state.editingPlot) {

await this.savePlotClick();

}

const plot = await PlotService.create({userId: this.props.currentUser.id});

plot.polygon = PlotPolygon(

this.state.ymaps,

{

geometry: [],

options: {

editorDrawingCursor: "crosshair",

fillColor: "#0000FF30",

strokeColor: "#0000FF",

strokeWidth: 5,

},

user: this.props.currentUser,

currentUser: this.props.currentUser,

plot: plot,

profileClick: this.props.profileClick,

editPlotClick: this.editPlotClick,

showPlotDetailsClick: this.showPlotDetailsClick,

}

);

this.mapRef.geoObjects.add(plot.polygon);

plot.polygon.editor.startDrawing();

this.setState(prevState => {

const currentUserPlots = [...prevState.currentUserPlots];

currentUserPlots[plot.id] = plot;

return {

editingPlot: plot,

currentUserPlots: currentUserPlots

};

});

};

editPlotClick = (plot) => {

if (plot.polygon.editor) {

plot.polygon.editor.startEditing();

}

this.setState({

editingPlot: plot,

}, () => {

this.setState({

modalWindow: (

<EditablePlot

plot={this.state.editingPlot}

updatePlot={this.updatePlot}

deletePlot={this.deletePlot}

onClose={() => this.setState({

modalWindow: undefined,

})}/>

),

});

});

};

showPlotDetailsClick = (plot) => {

this.setState({

modalWindow: (

<PlotDocument

plot={plot}

onClose={() => this.setState({

modalWindow: undefined,

})}/>

),

});

};

savePlotClick = async () => {

if(this.state.editingPlot) {

if (this.state.editingPlot.polygon.editor) {

this.state.editingPlot.polygon.editor.stopEditing();

}

if (this.state.editingPlot.polygon.geometry

.\_coordPath.\_coordinates[0].length > 3) {

await this.updatePlot(this.state.editingPlot);

} else {

this.mapRef.geoObjects.remove(this.state.editingPlot.polygon);

await PlotService.delete(this.state.editingPlot.id);

}

this.setState({

editingPlot: undefined,

})

}

};

updatePlot = async (plot) => {

if (!plot) {

return;

}

this.mapRef.geoObjects.remove(this.state.editingPlot.polygon);

const newPlot = await PlotService.update(new PlotModel(plot));

this.setState(prevState => {

const currentUserPlots = [...prevState.currentUserPlots];

currentUserPlots[newPlot.id] = newPlot;

return {

editingPlot: newPlot,

currentUserPlots: currentUserPlots,

};

}, () => {

const newPolygon = PlotPolygon(

this.state.ymaps,

{

geometry: this.state.editingPlot.polygon,

options: {

editorDrawingCursor: "crosshair",

fillColor: "#0000FF30",

strokeColor: "#0000FF",

strokeWidth: 5,

},

user: this.props.currentUser,

currentUser: this.props.currentUser,

plot: this.state.editingPlot,

profileClick: this.props.profileClick,

editPlotClick: this.editPlotClick,

showPlotDetailsClick: this.showPlotDetailsClick,

}

);

this.mapRef.geoObjects.add(newPolygon);

this.state.editingPlot.polygon = newPolygon;

console.log(this.state.editingPlot);

});

};

deletePlot = async (plotId) => {

this.mapRef.geoObjects.remove(this.state.editingPlot.polygon);

this.state.currentUserPlots[plotId] = null;

this.setState({

editingPlot: undefined,

});

await PlotService.delete(plotId);

};

componentDidMount() {

this.setCurrentPosition();

this.setPlots();

this.setUsers();

}

onMapLoad = (ymaps) => {

this.userPlots.forEach(plot => {

const newPolygon = PlotPolygon(

ymaps,

{

geometry: plot.polygon,

options: {

fillColor: "#FF000030",

strokeColor: "#770000",

strokeWidth: 5,

},

user: this.users[plot.userId],

currentUser: this.props.currentUser,

plot: plot,

profileClick: this.props.profileClick,

editPlotClick: this.editPlotClick,

showPlotDetailsClick: this.showPlotDetailsClick,

}

);

this.mapRef.geoObjects.add(newPolygon);

plot.polygon = newPolygon;

});

this.state.currentUserPlots.forEach(plot => {

const newPolygon = PlotPolygon(

ymaps,

{

geometry: plot.polygon,

options: {

editorDrawingCursor: "crosshair",

fillColor: "#0000FF30",

strokeColor: "#0000FF",

strokeWidth: 5,

},

user: this.props.currentUser,

currentUser: this.props.currentUser,

plot: plot,

profileClick: this.props.profileClick,

editPlotClick: this.editPlotClick,

showPlotDetailsClick: this.showPlotDetailsClick,

}

);

this.mapRef.geoObjects.add(newPolygon);

plot.polygon = newPolygon;

});

this.setState({ymaps: ymaps});

};

render() {

return (

<div className="map">

<YMaps query={{

lang: "en\_US",

apikey: "f8a4a7fe-f442-4a37-af5a-a4dec57c863f"}}>

{

this.state.userPosition && (

<Map instanceRef={ref => this.mapRef = ref}

defaultState={{center: this.state.userPosition, zoom: 12}}

options={{autoFitToViewport: 'always'}}

height="100%"

width="100%"

modules={[

"Polygon",

"geoObject.addon.editor",

"geoObject.addon.balloon",

"templateLayoutFactory",]}

onLoad={this.onMapLoad}>

<Button

options={{

maxWidth: 128,

selectOnClick: false,

}}

data={{content: 'Save plot'}}

onClick={this.savePlotClick}/>

<Button

options={{

maxWidth: 128,

selectOnClick: false,

}}

data={{content: 'Create new plot'}}

onClick={this.newPlotClick}/>

</Map>

)

}

{this.state.modalWindow && (this.state.modalWindow)}

</YMaps>

</div>

);

}

}

export {YandexMap}