ПРИЛОЖЕНИЕ Д

Листинг программного кода

СОДЕРЖАНИЕ

[Д.1 Листинг программного кода МК-подсистемы регистрации свободных парковочных мест 2](#_Toc516104556)

[Д.2 Листинг программного кода вычислительного хаба 12](#_Toc516104557)

[Д.3 Листинг программного кода серверной стороны 18](#_Toc516104558)

# Д.1 Листинг программного кода МК-подсистемы регистрации свободных парковочных мест

Весь исходный код МК-подсистемы регистрации свободных парковочных мест доступен в репозитории на GitHab: <https://github.com/kiryanenko/SmartParking-sensor>.

Листинг Д.1.1 – Файл «SmartParking-sensor.ino»

Driver \*driver;

ParkingPlace parkingPalces[PARKING\_PLACES\_COUNT];

Parameters &parameters = Parameters::instance();

SerialModule serialModule(new ReceiveMessageHandler(parkingPalces, PARKING\_PLACES\_COUNT));

Display display;

Payment \*payment;

void setup()

{

Serial.begin(9600);

setSyncProvider(RTC.get);

RadioModule \*radioModule = new RadioModule(PIN\_RESET\_LORA,

parameters.getSendingPeriod() / PARKING\_PLACES\_COUNT,

new RadioModuleHandler(parkingPalces, PARKING\_PLACES\_COUNT));

if (radioModule->init()) {

driver = radioModule;

} else {

delete radioModule;

driver = &serialModule;

}

SonarI2C::begin(PIN\_INT\_SONAR);

for (int i = 0; i < PARKING\_PLACES\_COUNT; ++i) {

parkingPalces[i].init(i + 1);

}

display.init();

payment = new Payment(&display, parkingPalces, driver);

payment->init();

delay(300);

driver->sendInit(parameters.getId(), parameters.getSensorSamplingPeriod(),

parameters.getSendingPeriod(), parameters.getDayCost(), parameters.getNightCost(),

parameters.getDayStartTime(), parameters.getNightStartTime()

);

}

void loop()

{

SonarI2C::doSonar(); // call every cycle, SonarI2C handles the spacing

static Timer sendingPeriod;

for (byte i = 0; i < PARKING\_PLACES\_COUNT; ++i) {

if (parkingPalces[i].monitor() || sendingPeriod.isFinished()) {

sendingPeriod.start(parameters.getSendingPeriod());

driver->sendParkingStatus(parameters.getId(), i + 1,

parkingPalces[i].isFree());

}

}

driver->handleRecieveMessages();

serialModule.handleRecieveMessages();

payment->exec();

static Timer sec;

if (sec.isFinished()) {

sec.start(1000);

display.drawClock();

}

delay(parameters.getSensorSamplingPeriod());

}

Листинг Д.1.2 – Файл «ParkingPlace.h»

#pragma once

#include <PCF8574\PCF8574.h>

#include <SonarI2C\SonarI2C.h>

#include "Timer.h"

class ParkingPlace

{

byte m\_id;

bool m\_isReserved;

bool m\_isFree;

Timer m\_reservationTimer;

PCF8574 \*m\_pcf;

SonarI2C \*m\_sensor;

public:

ParkingPlace() = default;

void init(const byte id);

// Считывает и возвращает данные с сенсора

// Возвращает true при изменении состояния

bool monitor();

bool isFree() const;

void reserve(uint32\_t time);

void cancelReservation();

private:

void setIsFree(const bool isFree);

void setReserve(const bool isReserve);

};

Листинг Д.1.3 – Файл «ParkingPlace.cpp»

#include "ParkingPlace.h"

#include "Parameters.h"

void ParkingPlace::init(const byte id)

{

m\_id = id;

m\_isReserved = false;

m\_pcf = new PCF8574;

const auto adress = (id < 8 ? 0x20 : 0x38) + id;

m\_pcf->begin(adress);

m\_sensor = new SonarI2C(adress, PIN\_TRIG, 4000);

m\_sensor->init();

m\_pcf->pinMode(PIN\_IS\_FREE, OUTPUT);

m\_pcf->pinMode(PIN\_IS\_BOOKED, OUTPUT);

monitor();

}

bool ParkingPlace::monitor()

{

const auto dist = m\_sensor->cm();

const auto isFree = dist > CAR\_DISTANCE || dist == 0;

const auto isChangeState = isFree != m\_isFree;

setIsFree(isFree);

if (m\_isReserved && m\_reservationTimer.isFinished()) {

setReserve(false);

}

return isChangeState;

}

void ParkingPlace::reserve(const uint32\_t time)

{

m\_reservationTimer.start(time \* 1000);

setReserve(true);

}

void ParkingPlace::cancelReservation()

{

setReserve(false);

}

void ParkingPlace::setReserve(const bool isReserve)

{

m\_isReserved = isReserve;

m\_pcf->digitalWrite(PIN\_IS\_BOOKED, m\_isReserved);

}

Листинг Д.1.4 – Файл «Driver.h»

#pragma once

#include <Arduino.h>

#include "AbstractReceiveMessageHandler.h"

class Driver

{

const uint8\_t type\_of\_send\_msg\_parking\_status = 'S';

const uint8\_t type\_of\_send\_msg\_init = 'I';

const uint8\_t type\_of\_send\_msg\_payment = 'P';

const uint8\_t type\_of\_recv\_msg\_set\_id = 'i';

const uint8\_t type\_of\_recv\_msg\_set\_sensor\_sampling\_period = 'a';

const uint8\_t type\_of\_recv\_msg\_set\_sending\_period = 'p';

const uint8\_t type\_of\_recv\_msg\_set\_time = 't';

const uint8\_t type\_of\_recv\_msg\_set\_settings = 's';

const uint8\_t type\_of\_recv\_msg\_reserve = 'r';

const uint8\_t type\_of\_recv\_msg\_cancel\_reservation = 'c';

const uint8\_t type\_of\_recv\_msg\_set\_day\_cost = 'q';

const uint8\_t type\_of\_recv\_msg\_set\_night\_cost = 'w';

const uint8\_t type\_of\_recv\_msg\_set\_day\_start\_time = 'd';

const uint8\_t type\_of\_recv\_msg\_set\_night\_start\_time = 'n';

AbstractReceiveMessageHandler \*m\_handler;

public:

explicit Driver(AbstractReceiveMessageHandler \*handler);

virtual ~Driver();

virtual bool init();

virtual bool send(const byte \*data, size\_t size) = 0;

virtual bool available() = 0;

virtual byte\* recv(size\_t &size) = 0;

void handleRecieveMessages();

void sendInit(uint32\_t id, uint16\_t samplingPeriod, uint16\_t sendingPeriod,

uint16\_t dayCost, uint16\_t nightCost, uint32\_t dayStartTime, uint32\_t nightStartTime);

void sendParkingStatus(uint32\_t id, uint8\_t parkingPlaceId, bool isFree);

void sendPayment(uint32\_t id, uint8\_t parkingPlaceId, uint32\_t time,

uint16\_t payment, uint16\_t totalCost);

protected:

const byte\* dataToSendInit(uint32\_t id, uint16\_t samplingPeriod,

uint16\_t sendingPeriod, uint16\_t dayCost, uint16\_t nightCost,

uint32\_t dayStartTime, uint32\_t nightStartTime, size\_t &bufSize) const;

const byte\* dataToSendParkingStatus(uint32\_t id, uint8\_t parkingPlaceId,

bool isFree, size\_t &bufSize) const;

const byte\* dataToSendPayment(uint32\_t id, uint8\_t parkingPlaceId, uint32\_t time,

uint16\_t payment, uint16\_t totalCost, size\_t& bufSize) const;

private:

void handleRecvMsgSetId(const byte \*msg, size\_t size);

void handleRecvMsgSetTime(const byte \*msg, size\_t size);

void handleRecvMsgSetSettings(const byte \*msg, size\_t size);

void handleRecvMsgReserve(const byte \*msg, size\_t size);

void handleRecvMsgCancelReservation(const byte \*msg, size\_t size);

};

Листинг Д.1.5 – Файл «Driver.cpp»

#include "Driver.h"

#include "MemUtils.h"

#include "Parameters.h"

void Driver::handleRecieveMessages()

{

while (available()) {

size\_t msgSize;

const byte \*msg = recv(msgSize);

if (msg) {

const auto type = msg[0];

const auto id = getReverseData<uint32\_t>(msg + sizeof type);

if (id == Parameters::instance().getId()) {

size\_t headerSize = sizeof id + sizeof type;

size\_t bodySize = msgSize - headerSize;

auto body = msg + headerSize;

if (type == type\_of\_recv\_msg\_set\_id) {

handleRecvMsgSetId(body, bodySize);

} else if (type == type\_of\_recv\_msg\_set\_time) {

handleRecvMsgSetTime(body, bodySize);

} else if (type == type\_of\_recv\_msg\_reserve) {

handleRecvMsgReserve(body, bodySize);

} else if (type == type\_of\_recv\_msg\_cancel\_reservation) {

handleRecvMsgCancelReservation(body, bodySize);

} else if (type == type\_of\_recv\_msg\_set\_settings) {

handleRecvMsgSetSettings(body, bodySize);

}

if (type == type\_of\_recv\_msg\_set\_settings) {

auto& parameters = Parameters::instance();

sendInit(parameters.getId(), parameters.getSensorSamplingPeriod(),

parameters.getSendingPeriod(), parameters.getDayCost(), parameters.getNightCost(),

parameters.getDayStartTime(), parameters.getNightStartTime()

);

}

}

delete[] msg;

}

}

}

void Driver::sendInit(uint32\_t id, uint16\_t samplingPeriod, uint16\_t sendingPeriod,

uint16\_t dayCost, uint16\_t nightCost, uint32\_t dayStartTime, uint32\_t nightStartTime)

{

size\_t bufSize;

const auto data = dataToSendInit(id, samplingPeriod, sendingPeriod,dayCost,

nightCost, dayStartTime, nightStartTime, bufSize);

send(data, bufSize);

delete[] data;

}

void Driver::sendParkingStatus(const uint32\_t id, const uint8\_t parkingPlaceId, bool isFree)

{

size\_t bufSize;

const auto data = dataToSendParkingStatus(id, parkingPlaceId, isFree, bufSize);

send(data, bufSize);

delete[] data;

}

void Driver::sendPayment(const uint32\_t id, const uint8\_t parkingPlaceId,

const uint32\_t time, const uint16\_t payment, const uint16\_t totalCost)

{

size\_t bufSize;

const auto data = dataToSendPayment(id, parkingPlaceId, time, payment, totalCost, bufSize);

send(data, bufSize);

delete[] data;

}

const byte\* Driver::dataToSendParkingStatus(const uint32\_t id, const uint8\_t parkingPlaceId,

const bool isFree, size\_t& bufSize) const

{

bufSize = 1 + 4 + 1 + 1;

const auto dataToSend = new byte[bufSize];

cpyReverseData(dataToSend, type\_of\_send\_msg\_parking\_status);

cpyReverseData(dataToSend + 1, id);

cpyReverseData(dataToSend + 1 + 4, parkingPlaceId);

cpyReverseData(dataToSend + 1 + 4 + 1, isFree);

return dataToSend;

}

void Driver::handleRecvMsgSetTime(const byte\* msg, size\_t size)

{

if (size == sizeof(uint16\_t)) {

const auto time = getReverseData<uint32\_t>(msg);

m\_handler->onSetTime(time);

}

}

void Driver::handleRecvMsgSetSettings(const byte \* msg, size\_t size)

{

if (size == 2 + 2 + 2 + 2 + 4 + 4) {

const auto samplingPeriod = getReverseData<uint16\_t>(msg);

const auto sandingPeriod = getReverseData<uint16\_t>(msg + sizeof(samplingPeriod));

const auto dayCost = getReverseData<uint16\_t>(msg + sizeof(samplingPeriod) +

sizeof(sandingPeriod));

const auto nightCost = getReverseData<uint16\_t>(msg + sizeof(samplingPeriod) +

sizeof(sandingPeriod) + sizeof(dayCost));

const auto dayStartTime = getReverseData<uint32\_t>(msg + sizeof(samplingPeriod) +

sizeof(sandingPeriod) + sizeof(dayCost) + sizeof(nightCost));

const auto nightStartTime = getReverseData<uint32\_t>(msg + sizeof(samplingPeriod) +

sizeof(sandingPeriod) + sizeof(dayCost) + sizeof(nightCost) + sizeof(dayStartTime));

m\_handler->onSetSettings(samplingPeriod, sandingPeriod, dayCost, nightCost,

dayStartTime, nightStartTime);

}

}

void Driver::handleRecvMsgReserve(const byte\* msg, size\_t size)

{

if (size == sizeof(uint8\_t) + sizeof(uint32\_t)) {

const auto placeId = getReverseData<uint8\_t>(msg);

const auto time = getReverseData<uint32\_t>(msg + sizeof(placeId));

m\_handler->onReserveMsg(placeId, time);

}

}

void Driver::handleRecvMsgCancelReservation(const byte\* msg, size\_t size)

{

if (size == sizeof(uint8\_t)) {

const auto placeId = getReverseData<uint8\_t>(msg);

m\_handler->onCancelReservationMsg(placeId);

}

}

Листинг Д.1.6 – Файл «AbstractReceiveMessageHandler.h»

#pragma once

#include <Arduino.h>

#include <Time.h>

class AbstractReceiveMessageHandler

{

public:

AbstractReceiveMessageHandler();

virtual ~AbstractReceiveMessageHandler();

virtual void onSetIdMsg(uint32\_t id) = 0;

virtual void onSetSamplingPeriodMsg(uint16\_t period) = 0;

virtual void onSetSendingPeriodMsg(uint16\_t period) = 0;

virtual void onSetTime(time\_t time) = 0;

virtual void onSetSettings(uint16\_t samplingPeriod, uint16\_t sendingPeriod,

uint16\_t dayCost, uint16\_t nightCost, uint32\_t dayStartTime, uint32\_t nightStartTime) = 0;

virtual void onReserveMsg(uint8\_t parkingPlaceId, uint32\_t time) = 0;

virtual void onCancelReservationMsg(uint8\_t parkingPlaceId) = 0;

virtual void onSetDayCost(uint16\_t cost) = 0;

virtual void onSetNightCost(uint16\_t cost) = 0;

virtual void onSetDayStartTime(uint16\_t time) = 0;

virtual void onSetNightStartTime(uint16\_t time) = 0;

};

Листинг Д.1.7 – Файл «ReceiveMessageHandler.h»

#pragma once

#include "AbstractReceiveMessageHandler.h"

#include "ParkingPlace.h"

class ReceiveMessageHandler :

public AbstractReceiveMessageHandler

{

ParkingPlace\* m\_parkingPlaces;

uint8\_t m\_parkingPlacesCount;

public:

ReceiveMessageHandler(ParkingPlace\* parkingPlaces, uint8\_t parkingPlacesCount);

void onSetIdMsg(uint32\_t id) override;

void onSetSamplingPeriodMsg(uint16\_t period) override;

void onSetSendingPeriodMsg(uint16\_t period) override;

void onSetTime(time\_t time) override;

void onReserveMsg(uint8\_t parkingPlaceId, uint32\_t time) override;

void onCancelReservationMsg(uint8\_t parkingPlaceId) override;

void onSetDayCost(uint16\_t cost) override;

void onSetNightCost(uint16\_t cost) override;

void onSetDayStartTime(uint16\_t time) override;

void onSetNightStartTime(uint16\_t time) override;

void onSetSettings(uint16\_t samplingPeriod, uint16\_t sendingPeriod, uint16\_t dayCost,

uint16\_t nightCost, uint32\_t dayStartTime, uint32\_t nightStartTime) override;

};

Листинг Д.1.8 – Файл «ReceiveMessageHandler.cpp»

#include "ReceiveMessageHandler.h"

#include <DS3232RTC.h>

#include "Parameters.h"

void ReceiveMessageHandler::onSetIdMsg(const uint32\_t id)

{

auto& params = Parameters::instance();

params.setId(id);

}

void ReceiveMessageHandler::onSetTime(const time\_t time)

{

RTC.set(time);

setSyncProvider(RTC.get);

}

void ReceiveMessageHandler::onReserveMsg(const uint8\_t parkingPlaceId, const uint32\_t time)

{

if (parkingPlaceId > 0 && parkingPlaceId <= m\_parkingPlacesCount) {

m\_parkingPlaces[parkingPlaceId - 1].reserve(time);

}

}

void ReceiveMessageHandler::onSetSettings(uint16\_t samplingPeriod, uint16\_t sendingPeriod,

uint16\_t dayCost, uint16\_t nightCost, uint32\_t dayStartTime, uint32\_t nightStartTime)

{

auto& params = Parameters::instance();

params.setSensorSamplingPeriod(samplingPeriod);

params.setSendingPeriod(sendingPeriod);

params.setDayCost(dayCost);

params.setNightCost(nightCost);

params.setDayStartTime(dayStartTime);

params.setNightStartTime(nightStartTime);

}

Листинг Д.1.9 – Файл «RadioModuleHandler.h»

#pragma once

#include "ReceiveMessageHandler.h"

class RadioModuleHandler : public ReceiveMessageHandler

{

public:

RadioModuleHandler(ParkingPlace \*parkingPlaces, uint8\_t parkingPlacesCount);

void onSetIdMsg(uint32\_t id) override {}

};

Листинг Д.1.10 – Файл «RadioModule.h»

#pragma once

#include "Driver.h"

#include <RadioHead\RH\_RF95.h>

class RadioModule : public Driver

{

RH\_RF95 m\_rf95;

int m\_pinResetLora;

int m\_timeout;

public:

RadioModule(int pinResetLora, int timeout, AbstractReceiveMessageHandler \*handler);

bool init() override;

bool send(const byte \*data, size\_t size) override;

bool available() override;

byte\* recv(size\_t &size) override;

private:

void reset() const;

};

Листинг Д.1.11 – Файл «Payment.h»

#pragma once

#include <Arduino.h>

#include <i2ckeypad.h>

#include "Display.h"

#include "Timer.h"

#include "ParkingPlace.h"

#include "Driver.h"

class Payment

{

i2ckeypad m\_keypad;

Display \*m\_display;

ParkingPlace \*m\_parkingPlaces;

Driver \*m\_driver;

String m\_inputStr;

Timer m\_timeout;

uint8\_t m\_parkingPlace;

uint16\_t m\_timeReserve;

float m\_totalCost;

enum State {START, ERROR, ENTER\_PARKING\_PLACE, ENTER\_TIME, PAYMENT, SUCCESS\_PAYMENT

} m\_state;

public:

Payment(Display\* display, ParkingPlace \*parkingPlaces, Driver \*driver);

void init();

void exec();

private:

void startState(char key);

void errorState(char key);

void successPaymentState(char key);

void inputState(char key, void(Payment::\* onSuccess)());

void onSuccessInputParkingPlace();

void onSuccessInputTime();

void onSuccessInputPayment();

void setState(State state);

float countingCost(time\_t time) const;

};

Листинг Д.1.12 – Файл «Payment.cpp»

#include "Payment.h"

#include <Time.h>

#include "Parameters.h"

void Payment::exec()

{

const auto key = m\_keypad.get\_key();

if (key != '\0') {

switch (m\_state) {

case START:

startState(key);

break;

case ENTER\_PARKING\_PLACE:

inputState(key, &onSuccessInputParkingPlace);

break;

case ENTER\_TIME:

inputState(key, &onSuccessInputTime);

break;

case PAYMENT:

inputState(key, &onSuccessInputPayment);

break;

case SUCCESS\_PAYMENT:

successPaymentState(key);

break;

case ERROR:

errorState(key);

break;

}

}

if (m\_state == ERROR && m\_timeout.isFinished()) {

setState(START);

m\_display->showStartPage();

} else if (m\_state != START && m\_state != ERROR && m\_timeout.isFinished()) {

setState(ERROR);

m\_display->showError(PSTR("Таймоут."));

}

}

void Payment::startState(const char key)

{

if (key == '1') {

setState(ENTER\_PARKING\_PLACE);

m\_display->showEnterParkingPlacePage();

}

}

void Payment::errorState(char key)

{

setState(START);

m\_display->showStartPage();

}

void Payment::inputState(const char key, void (Payment::\* onSuccess)())

{

if (key >= '0' && key <= '9') {

m\_inputStr += key;

m\_display->drawInput(m\_inputStr);

} else if (key == '\*') {

if (m\_inputStr.length() > 0) {

(void) (this->\*onSuccess)();

} else {

setState(ERROR);

m\_display->showError(PSTR("Данные не введены."));

}

} else if (key == '#') {

setState(ERROR);

m\_display->showError(PSTR("Платеж отменен."));

}

}

void Payment::onSuccessInputParkingPlace()

{

m\_parkingPlace = atoi(m\_inputStr.c\_str());

if (m\_parkingPlace < 1 || m\_parkingPlace > PARKING\_PLACES\_COUNT) {

setState(ERROR);

m\_display->showError(PSTR("Парковка не найдена."));

return;

}

setState(ENTER\_TIME);

m\_display->showEnterTimePage();

}

void Payment::onSuccessInputPayment()

{

const float payment = atoi(m\_inputStr.c\_str());

const auto change = payment - m\_totalCost;

if (change < 0) {

setState(ERROR);

m\_display->showError(PSTR("Недостаточно средств."));

return;

}

auto& params = Parameters::instance();

m\_driver->sendPayment(params.getId(), m\_parkingPlace, m\_timeReserve, payment,

m\_totalCost);

m\_parkingPlaces[m\_parkingPlace - 1].reserve(m\_timeReserve \* 60);

setState(SUCCESS\_PAYMENT);

m\_display->showSuccessPaymentPage(change);

}

float Payment::countingCost(const time\_t time) const

{

auto& params = Parameters::instance();

const uint16\_t now = hour() \* 3600 + minute() \* 60 + second(); // Секунды с начала дня

uint16\_t cost;

if (now > params.getDayStartTime() &&

(now < params.getNightStartTime() || params.getNightStartTime() <

params.getDayStartTime())) {

cost = params.getDayCost();

} else {

cost = params.getNightCost();

}

return time \* cost / 3600.0;

}

Листинг Д.1.13 – Файл «Parameters.h»

#pragma once

#include <Arduino-EEPROMEx\EEPROMex.h>

#define PARKING\_PLACES\_COUNT 1

#define PIN\_RESET\_LORA 9

#define PIN\_INT\_SONAR 3

#define KEYPAD\_I2C\_ADDR 0x20

#define KEYPAD\_ROWS 4

#define KEYPAD\_COLS 4

#define TIMEOUT 120000

#define MSG\_SHOW\_TIME 5000

class Parameters

{

uint32\_t m\_id; // Идентификатор устройства

uint16\_t m\_sensorSamplingPeriod; // Период опроса датчиков

uint16\_t m\_sendingPeriod; // Период отправки сообщений

uint16\_t m\_dayCost; // Дневная тариф

uint16\_t m\_nightCost; // Ночная ториф

uint32\_t m\_dayStartTime; // Время (количество сек с 00:00) начала дневного тарифа

uint32\_t m\_nightStartTime; // Время (количество сек с 00:00) начала ночного тарифа

const int adress\_id = 0;

const int adress\_sensor\_sampling\_period = adress\_id + sizeof(m\_id);

const int adress\_sending\_period = adress\_sensor\_sampling\_period +

sizeof(m\_sensorSamplingPeriod);

const int adress\_day\_cost = adress\_sending\_period + sizeof(m\_sendingPeriod);

const int adress\_night\_cost = adress\_day\_cost + sizeof(m\_dayCost);

const int adress\_day\_start\_time = adress\_night\_cost + sizeof(m\_nightCost);

const int adress\_night\_start\_time = adress\_day\_start\_time + sizeof(m\_dayStartTime);

public:

static Parameters& instance();

uint32\_t getId() const;

uint16\_t getSensorSamplingPeriod() const;

uint16\_t getSendingPeriod() const;

uint16\_t getDayCost() const;

uint16\_t getNightCost() const;

uint32\_t getDayStartTime() const;

uint32\_t getNightStartTime() const;

void setId(uint32\_t id);

void setSendingPeriod(uint16\_t sendingPeriod);

void setSensorSamplingPeriod(uint16\_t samplingPeriod);

void setDayCost(uint16\_t cost);

void setNightCost(uint16\_t cost);

void setDayStartTime(uint32\_t time);

void setNightStartTime(uint32\_t time);

private:

Parameters();

Parameters(const Parameters& root);

Parameters& operator=(const Parameters&);

template <typename T>

void save(T &field, const T &data, int adress);

};

# Д.2 Листинг программного кода вычислительного хаба

Весь программный код вычислительного хаба доступен в репозитории на GitHab: <https://github.com/kiryanenko/SmartParking-Transceiver>.

Листинг Д.2.1 – Файл «Worker.h»

#ifndef WORKERTHREAD\_H

#define WORKERTHREAD\_H

#include <QSettings>

#include "Driver.h"

class Worker : public QObject

{

Q\_OBJECT

Driver \*m\_driver;

public:

Worker(QSettings \*settngs, QObject \*parent = 0);

public slots:

void **run**();

};

#endif // WORKERTHREAD\_H

Листинг Д.2.2 – Файл «Worker.cpp»

#include "LoRaConnection.h"

#include "Worker.h"

#include <QSerialPortInfo>

#include <QDebug>

#include "SerialConnection.h"

#include "ReceiveMessageHandler.h"

Worker::**Worker**(QSettings \*settings, QObject \*parent) : QObject(parent)

{

QList<quint32> sensors;

auto sensorsInSettings = settings->value("sensors").toList();

for (QVariant value : sensorsInSettings) {

sensors << value.toLongLong();

}

QList<Server\*> servers;

for(QJsonValue rec : QJsonDocument::fromJson(

settings->value("servers").toByteArray()).array()) {

servers << new Server(rec["host"].toString(), rec["mqtt\_port"].toInt(),

rec["mqtt\_username"].toString(), rec["mqtt\_password"].toString(),

rec["login"].toString(), rec["password"].toString(),

sensors, new AbstractReceiveMessageHandler(parent), parent);

}

QString driverType = settings->value("driver").toString().toLower();

if (driverType == "rfm95" || driverType == "lora") {

m\_driver = new LoRaConnection(sensors, settings->value("frequency").toInt(),

settings->value("lora\_timeout").toInt());

} else {

QString availablePorts = "Available ports: ";

for (QSerialPortInfo info : QSerialPortInfo().availablePorts()) {

availablePorts += info.portName() + ' ';

}

qDebug() << availablePorts;

m\_driver = new SerialConnection(sensors,

settings->value("serial\_port").toString(), settings->value("baud\_rate").toInt(),

new AbstractReceiveMessageHandler(parent), parent);

}

auto \*handler = new ReceiveMessageHandler(servers, m\_driver, this);

m\_driver->setHandler(handler);

for (Server \*serv : servers) {

serv->setHandler(handler);

}

}

void Worker::**run**()

{

m\_driver->handleRecieveMessages();

}

Листинг Д.2.3 – Файл «Driver.h»

#pragma once

#include <QtCore>

#include <QByteArray>

#include <QDataStream>

#include <QList>

#include "AbstractReceiveMessageHandler.h"

class Driver : public QObject

{

Q\_OBJECT

const quint8 type\_of\_recv\_msg\_parking\_status = 'S';

const quint8 type\_of\_recv\_msg\_init = 'I';

const quint8 type\_of\_recv\_msg\_payment = 'P';

const quint8 type\_of\_send\_msg\_set\_id = 'i';

const quint8 type\_of\_send\_msg\_set\_sensor\_sampling\_period = 'a';

const quint8 type\_of\_send\_msg\_set\_sending\_period = 'p';

const quint8 type\_of\_send\_msg\_set\_settings = 's';

const quint8 type\_of\_send\_msg\_reserve = 'r';

const quint8 type\_of\_send\_msg\_cancel\_reservation = 'c';

AbstractReceiveMessageHandler \*m\_handler;

QList<quint32> m\_sensors;

public:

explicit Driver(QList<quint32> &sensors,

AbstractReceiveMessageHandler \*handler = new AbstractReceiveMessageHandler(),

QObject \*parent = 0);

virtual ~***Driver***();

virtual bool ***send***(QByteArray data) = 0;

virtual bool ***available***() = 0;

virtual QByteArray ***recv***() = 0;

void **setHandler**(AbstractReceiveMessageHandler \*handler);

void **handleRecieveMessages**();

void **sendSetId**(quint32 sensorId, quint32 newId);

void **sendSetSensorSamplingPeriod**(quint32 sensorId, quint16 samplingPeriod);

void **sendSetSendingPeriod**(quint32 sensorId, quint16 sendingPeriod);

void **sendSetSettings**(quint32 sensorId, quint16 samplingPeriod,

quint16 sendingPeriod, quint16 dayCost, quint16 nightCost,

quint32 dayStartTime, quint32 nightStartTime);

void **sendReserve**(quint32 sensorId, quint8 parkingPlaceId, quint32 time);

void **sendCancelReservation**(quint32 sensorId, quint8 parkingPlaceId);

private:

void **handleRecvParkingState**(quint32 id, QDataStream &stream);

void **handleRecvInit**(quint32 id, QDataStream &stream);

void **handleRecvPayment**(quint32 id, QDataStream &stream);

};

Листинг Д.2.4 – Файл «Driver.cpp»

#include "Driver.h"

void Driver::**handleRecieveMessages**()

{

while (*available*()) {

auto msg = *recv*();

if (!msg.isEmpty()) {

m\_handler->*onRecv*(msg);

QDataStream stream(&msg, QIODevice::ReadOnly);

quint8 type;

quint32 id;

stream >> type >> id;

if (m\_sensors.indexOf(id) != -1) {

if (type == type\_of\_recv\_msg\_parking\_status) {

handleRecvParkingState(id, stream);

} else if (type == type\_of\_recv\_msg\_init) {

handleRecvInit(id, stream);

} else if (type == type\_of\_recv\_msg\_payment) {

handleRecvPayment(id, stream);

} else {

qCritical() << "[WARN] Unknown type:" << type;

}

} else {

qWarning() << "[WARN] Unknown sensor ID:" << id;

}

}

}

}

void Driver::**sendSetSettings**(quint32 sensorId, quint16 samplingPeriod,

quint16 sendingPeriod, quint16 dayCost, quint16 nightCost,

quint32 dayStartTime, quint32 nightStartTime)

{

QByteArray dataToSend;

QDataStream stream(&dataToSend, QIODevice::ReadWrite);

stream << type\_of\_send\_msg\_set\_settings << sensorId << samplingPeriod

<< sendingPeriod << dayCost << nightCost << dayStartTime << nightStartTime;

*send*(dataToSend);

}

void Driver::**sendReserve**(quint32 sensorId, quint8 parkingPlaceId, quint32 time)

{

QByteArray dataToSend;

QDataStream stream(&dataToSend, QIODevice::ReadWrite);

stream << type\_of\_send\_msg\_reserve << sensorId << parkingPlaceId << time;

*send*(dataToSend);

}

void Driver::**handleRecvParkingState**(quint32 id, QDataStream &stream)

{

quint8 place;

bool isFree;

stream >> place >> isFree;

m\_handler->*onParkingStatus*(id, place, isFree);

}

void Driver::**handleRecvInit**(quint32 id, QDataStream &stream)

{

quint16 samplingPeriod, sendingPeriod;

quint16 dayCost, nightCost;

quint32 dayStartTime, nightStartTime;

stream >> samplingPeriod >> sendingPeriod >> dayCost >> nightCost >> dayStartTime

>> nightStartTime;

m\_handler->*onInit*(id, samplingPeriod, sendingPeriod, dayCost, nightCost,

dayStartTime, nightStartTime);

}

void Driver::**handleRecvPayment**(quint32 id, QDataStream &stream)

{

quint8 place;

quint32 time;

quint16 payment, totalCost;

stream >> place >> time >> payment >> totalCost;

m\_handler->*onPayment*(id, place, time, payment, totalCost);

}

Листинг Д.2.5 – Файл «LoRaConnection.h»

#ifndef LORACONNECTION\_H

#define LORACONNECTION\_H

#include "Driver.h"

#include <bcm2835.h>

#include <RH\_RF95.h>

#include <RadioHead.h>

class LoRaConnection : public Driver

{

Q\_OBJECT

RH\_RF95 m\_rf95;

public:

explicit LoRaConnection(QList<quint32> &sensors, int frequency, uint16\_t timeout,

AbstractReceiveMessageHandler \*handler = new AbstractReceiveMessageHandler());

bool send(QByteArray data) Q\_DECL\_OVERRIDE;

bool available() Q\_DECL\_OVERRIDE;

QByteArray recv() Q\_DECL\_OVERRIDE;

};

#endif // LORACONNECTION\_H

Листинг Д.2.6 – Файл «ReceiveMessageHandler.h»

#pragma once

#include "AbstractReceiveMessageHandler.h"

#include "Server.h"

#include "Driver.h"

class ReceiveMessageHandler : public AbstractReceiveMessageHandler

{

Q\_OBJECT

QList<Server\*> m\_servers;

Driver \*m\_driver;

public:

ReceiveMessageHandler(QList<Server\*> &servers, Driver \*driver,

QObject \*parent = 0);

void ***onRecv***(QByteArray data) override;

void ***onParkingStatus***(quint32 id, quint8 place, bool isFree) override;

void ***onInit***(quint32 id, quint16 samplingPeriod, quint16 sendingPeriod,

quint16 dayCost, quint16 nightCost,

quint32 dayStartTime, quint32 nightStartTime) override;

void ***onPayment***(quint32 id, quint8 place, quint32 bookedTime,

quint16 payment, quint16 totalCost) override;

void ***onBook***(quint32 sensor, quint8 place, quint32 bookedTime) override;

void ***onSetSensorSettings***(quint32 sensor, quint16 samplingPeriod,

quint16 sendingPeriod, quint16 dayCost, quint16 nightCost,

quint32 dayStartTime, quint32 nightStartTime) override;

};

Листинг Д.2.7 – Файл «Server.h»

#ifndef SERVER\_H

#define SERVER\_H

#include <QObject>

#include <QMqttClient>

#include "AbstractReceiveMessageHandler.h"

class Server : QObject

{

Q\_OBJECT

QMqttClient \*m\_mqtt;

QString m\_host;

quint16 m\_mqttPort;

QString m\_mqttUsername;

QString m\_mqttPwd;

QString m\_login;

QString m\_pwd;

AbstractReceiveMessageHandler \*m\_handler;

QList<quint32> m\_sensors;

public:

Server(QString host, quint16 mqtt\_port, QString mqttUsername,

QString mqttPwd, QString login, QString pwd, QList<quint32> sensors,

AbstractReceiveMessageHandler \*handler = new AbstractReceiveMessageHandler(), QObject \*parent = 0);

void **setHandler**(AbstractReceiveMessageHandler \*handler);

void **sendParkingStatus**(qint64 id, quint8 place, bool isFree);

void **sendInit**(qint64 id, quint16 samplingPeriod, quint16 sendingPeriod,

quint16 dayCost, quint16 nightCost, qint64 dayStartTime, qint64 nightStartTime);

void **sendPayment**(qint64 id, quint8 place, qint64 bookedTime,

quint16 payment, quint16 totalCost);

static QString **bookTopic**(quint32 sensor);

static QString **settingsTopic**(quint32 sensor);

public slots:

void **onConnected**();

void **onDisconnected**();

void **onMsgRecv**(QByteArray msg, QMqttTopicName topic);

private:

void **mqttConnect**();

bool **mqttCheckConnection**();

void **handleRecvBook**(quint32 sensor, QByteArray data);

void **handleRecvSettings**(quint32 sensor, QByteArray data);

};

#endif // SERVER\_H

Листинг Д.2.8 – Файл «Server.cpp»

#include "Server.h"

#include <QJsonObject>

#include <QJsonDocument>

Server::**Server**(QString host, quint16 mqttPort, QString mqttUsername,

QString mqttPwd, QString login, QString pwd, QList<quint32> sensors,

AbstractReceiveMessageHandler \*handler, QObject \*parent) :

m\_host(host), m\_mqttPort(mqttPort), m\_mqttUsername(mqttUsername),

m\_mqttPwd(mqttPwd), m\_login(login), m\_pwd(pwd), m\_sensors(sensors), m\_handler(handler), QObject(parent)

{

m\_mqtt = new QMqttClient(this);

m\_mqtt->setHostname(host);

m\_mqtt->setPort(mqttPort);

m\_mqtt->setUsername(mqttUsername);

m\_mqtt->setPassword(mqttPwd);

connect(m\_mqtt, SIGNAL(connected()), this, SLOT(onConnected()));

connect(m\_mqtt, SIGNAL(disconnected()), this, SLOT(onDisconnected()));

mqttConnect();

}

void Server::**sendParkingStatus**(qint64 id, quint8 place, bool isFree)

{

mqttCheckConnection();

QJsonObject response = {

{"login", m\_login},

{"password", m\_pwd},

{"sensor", id},

{"place\_id", place},

{"free", isFree}

};

m\_mqtt->publish(QMqttTopicName("status"), QJsonDocument(response).toJson());

}

void Server::**sendInit**(qint64 id, quint16 samplingPeriod, quint16 sendingPeriod,

quint16 dayCost, quint16 nightCost, qint64 dayStartTime, qint64 nightStartTime)

{

mqttCheckConnection();

QJsonObject response = {

{"login", m\_login},

{"password", m\_pwd},

{"sensor", id},

{"sampling\_period", samplingPeriod},

{"sending\_period", sendingPeriod},

{"day\_cost", dayCost},

{"night\_cost", nightCost},

{"day\_start\_time", dayStartTime},

{"night\_start\_time", nightStartTime}

};

m\_mqtt->publish(QMqttTopicName("init"), QJsonDocument(response).toJson());

}

void Server::**sendPayment**(qint64 id, quint8 place, qint64 bookedTime,

quint16 payment, quint16 totalCost)

{

mqttCheckConnection();

QJsonObject response = {

{"login", m\_login},

{"password", m\_pwd},

{"sensor", id},

{"place\_id", place},

{"booked\_time", bookedTime},

{"payment", payment},

{"total\_cost", totalCost}

};

m\_mqtt->publish(QMqttTopicName("payment"),QJsonDocument(response).toJson());

}

void Server::**onConnected**()

{

for (auto sensor : m\_sensors) {

auto subscription = m\_mqtt->subscribe(QMqttTopicFilter(bookTopic(sensor)));

subscription = m\_mqtt->subscribe(QMqttTopicFilter(settingsTopic(sensor)));

}

connect(m\_mqtt, SIGNAL(messageReceived(QByteArray, QMqttTopicName)),

this, SLOT(onMsgRecv(QByteArray, QMqttTopicName)));

}

void Server::**onMsgRecv**(QByteArray msg, QMqttTopicName topic)

{

try {

for (auto sensor : m\_sensors) {

if (topic.name() == bookTopic(sensor)) {

handleRecvBook(sensor, msg);

return;

}

if (topic.name() == settingsTopic(sensor)) {

handleRecvSettings(sensor, msg);

return;

}

}

} catch (...) {

qCritical() << "[ERROR] Error at handle received mqtt msg";

}

}

void Server::**handleRecvBook**(quint32 sensor, QByteArray data)

{

auto json = QJsonDocument::fromJson(data);

m\_handler->*onBook*(sensor, json["place\_id"].toInt(), json["booking\_time"].toInt());

}

void Server::**handleRecvSettings**(quint32 sensor, QByteArray data)

{

auto json = QJsonDocument::fromJson(data);

m\_handler->*onSetSensorSettings*(sensor,json["sampling\_period"].toInt(),

json["sending\_period"].toInt(),json["day\_cost"].toInt(), json["night\_cost"].toInt(), json["day\_start\_time"].toInt(), json["night\_start\_time"].toInt());

}

# Д.3 Листинг программного кода серверной стороны

Весь программный код серверной стороны доступен в репозитории на GitHab: <https://github.com/kiryanenko/SmartParking-Web>.

Листинг Д.3.1 – Файл «/app/channels/map\_channel.rb»

**require** './app/utils/hash'

**class** MapChannel **<** **ApplicationCable::Channel**

**def** subscribed

client\_id = connection.id

client = MapClient.**new** client\_id, params

stream\_from client.stream

stream\_from client.square.stream

client.send\_parkings

MapService.instance.add\_client client

**end**

**def** unsubscribed

MapService.instance.remove\_client connection.id

**end**

**def** **receive(**data**)**

data.recursive\_transform\_keys! **{|**k**|** k.to\_sym **}**

client = MapService.instance.get\_client connection.id

stop\_all\_streams

stream\_from client.stream

MapService.instance.update\_client client, data

stream\_from client.square.stream

**end**

**end**

Листинг Д.3.2 – Файл «/app/services/map\_service.rb»

**class** MapService

**include** **Singleton**

**def** initialize

**@map\_clients** = **Hash**.**new**

**@squares** = **Hash**.**new**

**@squares\_m** = **Mutex**.**new**

run

**end**

**def** add\_square**(**square**)**

**@squares\_m**.synchronize **do**

**if** **@squares**.has\_key? square.stream

sq = **@squares[**square.stream**]**

sq**[**:**count]** **+**= 1

**else**

**@squares[**square.stream**]** = **{**square: square, **count**: 1**}**

**end**

**end**

**end**

**def** remove\_square**(**square**)**

**@squares\_m**.synchronize **do**

sq = **@squares[**square.stream**]**

sq**[**:**count]** **-**= 1

**if** sq**[**:**count]** **<**= 0

**@squares**.**delete** square.stream

**end**

**end**

**end**

private

**def** run

**Thread**.**new** **do**

**loop** **do**

before = **Time**.now

**begin**

values = **[]**

**@squares\_m**.synchronize **{** values = **@squares**.values.dup **}**

values.**each** **do** **|**value**|**

value**[:square]**.broadcast

**end**

ParkingPlace.unset\_changed

**rescue** **Exception** **=>** e

Rails.logger.error e.message

**end**

sleep\_time = Rails.configuration.map\_sending\_period **-** **(Time**.now **-** before**)**

**sleep** sleep\_time **if** sleep\_time **>** 0

**end**

**end**

**end**

**end**

Листинг Д.3.3 – Файл «/app/models/map\_square.rb»

**require** './app/utils/numeric'

*# Для оптимизации карта была поделена на пересекающиеся квадраты.*

*# Таким образом, было ограничено количество запросов к БД.*

**class** MapSquare

attr\_reader **:coord**, **:radius**, **:cost**, **:with\_disabled**, **:only\_free**, **:can\_book**

**def** initialize**(**params**)**

r\_min = Rails.configuration.min\_map\_square\_side **/** 2

**@radius** = r\_min

**if** params**[:radius]** **>** r\_min

scale = **Math**.log2**(**params**[:radius]** **/** r\_min**)**.ceil

**@radius** = r\_min **\*** **(**2 **\*\*** scale**)**

**end**

*# Поиск ближайшей точки для центра*

n\_lat = **(**params**[:coord][:lat]** **/** **@radius)**.floor

n\_lng = **(**params**[:coord][:lng]** **/** **@radius)**.floor

**@coord** = **{**

lat: params**[:coord][:lat]**.near**(**@radius **\*** n\_lat, **@radius** **\*** **(**n\_lat **+** 1**))**,

lng: params**[:coord][:lng]**.near**(**@radius **\*** n\_lng, **@radius** **\*** **(**n\_lng **+** 1**))**

**}**

**@cost** = params**[:cost]** **||** **-**1

**@only\_free** = params**[:only\_free]**.**nil**? ? **true** : params**[:only\_free]**

**@can\_book** = params**[:can\_book]** **||** **false**

**@with\_disabled** = params**[:with\_disabled]** **||** **false**

**end**

**def** parkings

params = **{}**

**if** **@only\_free**

params**[:free]** = **true**

params**[:booked]** = **false**

params**[:connected]** = **true**

**end**

params**[:can\_book]** = **true** **if** **@can\_book**

params**[:for\_disabled]** = **false** **unless** **@with\_disabled**

Parking.response\_parkings\_at\_location **@coord**, **@radius**, **@cost**, params

**End**

**def** parkings\_cache

Rails.cache.fetch**(**stream, expires\_in: Rails.configuration.min\_map\_sending\_period**)** **do**

parkings

**end**

**end**

**def** broadcast

ActionCable.server.broadcast stream, parkings

**end**

**def** stream

"square\_R:#{@radius}\_LAT:#{@coord[:lat]}\_LNG:#{@coord[:lng]}\_COST:#{@cost}\_FREE:#{@only\_free}\_BOOK:#{@can\_book}\_DISABLED:#{@with\_disabled}"

**end**

**end**

Листинг Д.3.4 – Файл «/app/models/parking.rb»

**class** Parking **<** ApplicationRecord

**belongs\_to** **:user**

**has\_many** **:parking\_places**, dependent: :**destroy**

before\_validation **:ensure\_times\_both\_nil**

validates **:title**, **:area**, presence: **true**

scope **:find\_for\_user**, **->(**id, user**)** **{** find\_by! id: id, user: user **}**

scope **:user\_parkings**, **->(**user**)** **{** where**(**user: user**)**.order**(:id)** **}**

scope **:parkings\_at\_location**, **->(**coord, radius, cost = **-**1, params = **{})** **do**

res = where**(**"ST\_Intersects(ST\_GeographyFromText('SRID=4326;POLYGON((

:area\_lat1 :area\_lng1,

:area\_lat2 :area\_lng2,

:area\_lat3 :area\_lng3,

:area\_lat4 :area\_lng4,

:area\_lat1 :area\_lng1))'), parkings.area)",

area\_lat1: coord**[:lat]** **-** radius, area\_lng1: coord**[:lng]** **-** radius,

area\_lat2: coord**[:lat]** **-** radius, area\_lng2: coord**[:lng]** **+** radius,

area\_lat3: coord**[:lat]** **+** radius, area\_lng3: coord**[:lng]** **+** radius,

area\_lat4: coord**[:lat]** **+** radius, area\_lng4: coord**[:lng]** **-** radius,

**)**

res = res.where**(**'cost <= ?', cost**)** **if** cost **>** **-**1

res = res.joins**(:parking\_places)**.where**(**parking\_places: params**)**.distinct **if**

params.any?

res

**end**

**end**

Листинг Д.3.5 – Файл «/app/services/mqtt\_service.rb»

**class** MQTTService

**include** **Singleton**

**def** initialize

**begin**

connect

**rescue** **Exception** **=>** e

Rails.logger.error 'ERROR! Can not connect to MQTT: ' **+** e.message

Rails.logger.error e.backtrace

**end**

**end**

**def** connect

**@mqtt** = **MQTT::Client**.connect**(**ENV**[**"MQTT\_URI"**]** **||** ENV**[**"CLOUDMQTT\_URL"**]** **||**

'mqtt://0.0.0.0'**)**

**@mqtt**.subscribe 'init'

**@mqtt**.subscribe 'status'

**@mqtt**.subscribe 'payment'

run

**end**

**def** set\_settings**(**sensor**)**

**begin**

**@mqtt**.publish**(**"sensor\_#{sensor.id}-settings", JSON.generate**({**

sampling\_period: sensor.sampling\_period, sending\_period: sensor.sending\_period,

day\_cost: sensor.day\_cost, night\_cost: sensor.night\_cost,

day\_start\_time: sensor.day\_start\_time, night\_start\_time: sensor.night\_start\_time**}))**

**rescue** **Exception** **=>** e

Rails.logger.error e.message

**end**

**end**

**def** book**(**parking\_place, booking\_time**)**

**begin**

**@mqtt**.publish**(**"sensor\_#{parking\_place.sensor.id}-book", JSON.generate**({**

place\_id: parking\_place.place\_id, booking\_time: booking\_time**}))**

**rescue** **Exception** **=>** e

Rails.logger.error e.message

**end**

**end**

private

**def** run

**Thread**.**new** **do**

**@mqtt**.get **do** **|**topic,message**|**

**begin**

data = JSON.parse**(**message**)**.transform\_keys! **{|**k**|** k.to\_sym **}**

user = User.authenticate! data**[:login]**, data**[:password]**

**case** topic

**when** 'init'

sensor = Sensor.find\_for\_user data**[:sensor]**, user

sensor.**update(**

sampling\_period: data**[:sampling\_period]**,

sending\_period: data**[:sending\_period]**,

day\_cost: data**[:day\_cost]**, night\_cost: data**[:night\_cost]**,

day\_start\_time: data**[:day\_start\_time]**,

night\_start\_time: data**[:night\_start\_time]**

**)**

**when** 'status'

place = ParkingPlace.find\_by\_place\_id\_and\_user data**[:place\_id]**,

data**[:sensor]**, user

ParkingState.set\_state place, data**[:free]**

**when** 'payment'

place = ParkingPlace.find\_by\_place\_id\_and\_user data**[:place\_id]**,

data**[:sensor]**, user

order = Order.payment**(nil**, place, data**[:booked\_time]**, data**[:payment]**,

data**[:cost])**

order.**save!**

**end**

**rescue** **Exception** **=>** e

Rails.logger.error e.message

**end**

**end**

**end**

**end**

**end**

Листинг Д.3.6 – Файл «/app/assets/javascripts/main.js»

*//= require channels/map*

*//= require parking*

*//= require parking\_place.js.erb*

'use strict';

class MainMap {

constructor(map, parkings = [], parkingPlaces = []) {

**this**.map = **new** google.maps.Map(map, {

zoom: 13,

center: MAP\_CENTER

});

**this**.cluster = **new** MarkerClusterer(**this**.map, [],{imagePath: '/assets/images/m'});

**this**.parkings = **new** Map();

**this**.parkingPlaces = **new** Map();

**this**.update(parkings, parkingPlaces);

**this**.channel = **new** MapChannel(MAP\_CENTER, **this**.getRadius(),

**this**.update.bind(**this**));

**this**.map.bounds\_changed = **this**.sendSetParams.bind(**this**);

**this**.costField.oninput = **this**.onChangeCostField.bind(**this**);

**this**.costRangeField.oninput = **this**.onChangeCostRangeField.bind(**this**);

}

update(parkings, parkingPlaces) {

**this**.updateData(**this**.parkings, parkings, **this**.createParking.bind(**this**));

**this**.updateData(**this**.parkingPlaces, parkingPlaces,

**this**.createParkingPlace.bind(**this**));

}

updateData(data, newData, create) {

let newDataMap = **new** Map();

newData.forEach((el) => { newDataMap.**set**(el.id, el) });

**for** (let id of data.keys()) {

let current = data.**get**(id);

**if** (!newDataMap.has(id)) {

current.remove();

data.**delete**(id);

**continue**;

}

current.properties = newDataMap.**get**(id);

}

newData.forEach((newEl) => {

**if** (!data.has(newEl.id)) {

data.**set**(newEl.id, create(newEl));

}

});

}

createParking(parking) {

**return** **new** Parking(**this**.map, parking.area, parking);

}

createParkingPlace(place) {

**return** **new** ParkingPlace(**this**.map, place.coord, place, **false**, **this**.cluster);

}

getRadius() {

let bounds = **this**.map.getBounds();

let ne = bounds.getNorthEast();

let sw = bounds.getSouthWest();

**return** Math.sqrt((ne.lat() - sw.lat()) \* (ne.lat() - sw.lat()) +

(ne.lng() - sw.lng()) \* (ne.lng() - sw.lng())) / 2;

}

sendSetParams() {

**this**.channel.setParams(**this**.map.getCenter(), **this**.getRadius(),

parseFloat(**this**.costField.value), **this**.onlyFreeField.checked,

**this**.canBookField.checked, **this**.withDisabledField.checked

);

}

}

Листинг Д.3.7 – Файл «/app/assets/javascripts/channels/map.js»

*//= require cable*

'use strict';

class MapChannel {

constructor(coord, radius, onRecv) {

**this**.connection = App.cable.subscriptions.create({

channel: "MapChannel",

coord: coord,

radius: radius

}, {

received: (data) => {

console.log(data);

let parking\_places = data.reduce((res, parking) => {

parking.parking\_places.forEach((place) => {

place.parking = parking;

res.push(place);

});

**return** res;

}, []);

onRecv(data, parking\_places);

}

});

}

setParams(coord, radius, cost, onlyFree, canBook, withDisabled) {

**this**.connection.send({

coord: coord,

radius: radius,

cost: cost,

only\_free: onlyFree,

can\_book: canBook,

with\_disabled: withDisabled});

}

}