ПРИЛОЖЕНИЕ Д

Листинг программного кода

СОДЕРЖАНИЕ

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 тариф
    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_f
ree}_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});
    }
}
```