ПРИЛОЖЕНИЕ В

Тестирование системы

СОДЕРЖАНИЕ

В.1 Тестирование МКПРСПМ 2
В.1.1 Описание отладочной среды 2
В.1.2 Описание отладочного макета 2
В.1.2.1 Структура отладочного макета 2
В.1.2.2 Arduino Uno R3 3
В.1.2.3 LoRa Dragino Sheild 5
В.1.2.4 Радиомодуль Dragino LoRa BEE 6
В.1.3 Тестирование отладочного макета 7
В.1.3.1 Тестирование случая отсутствия ТС на парковочном месте 7
В.1.3.2 Тестирование случая присутствия ТС на парковочном месте 8
В.1.3.3 Тестирование бронирования парковочного места 9
В.1.3.4 Тестирование оплаты парковочного места 10
В.1.3.5 Результаты тестирования ПО МКПРСПМ 12
В.2 Тестирование вычислительного хаба 13
В.3 Тестирование сервера 15
В.3.1 Стратегии тестирования 15
В.3.2 Автоматизированное тестирование 15
В.3.3 Нагрузочное тестирование 15
В.4 Комплексное тестирование системы 19
В.4.1 Тестирование изменения состояния парковочного места 19
В.4.2 Тестирование онлайн бронирования 21
В.4.3 Тестирование бронирования через терминал оплаты 22
В.4.4 Тестирование удаленной настройки датчика 23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25

В.1 Тестирование МКПРСПМ

В.1.1 Описание отладочной среды

Программное обеспечение создавалось и отлаживалось в программной среде разработки Visual Studio версии 2017. Данная IDE очень удобна для разработки программного обеспечения под Arduino. Для разработки требуется установить специально плагин – «vMicro». Размер программного кода после компиляции составил 23 966 байт.

В.1.2 Описание отладочного макета

В.1.2.1 Структура отладочного макета

Тестирование разработанной системы проводилось на отладочной плате Arduino Uno R3 с использованием реальных электронных компонентов. Внешний вид отладочного макета приведен на рисунке В.1.1.

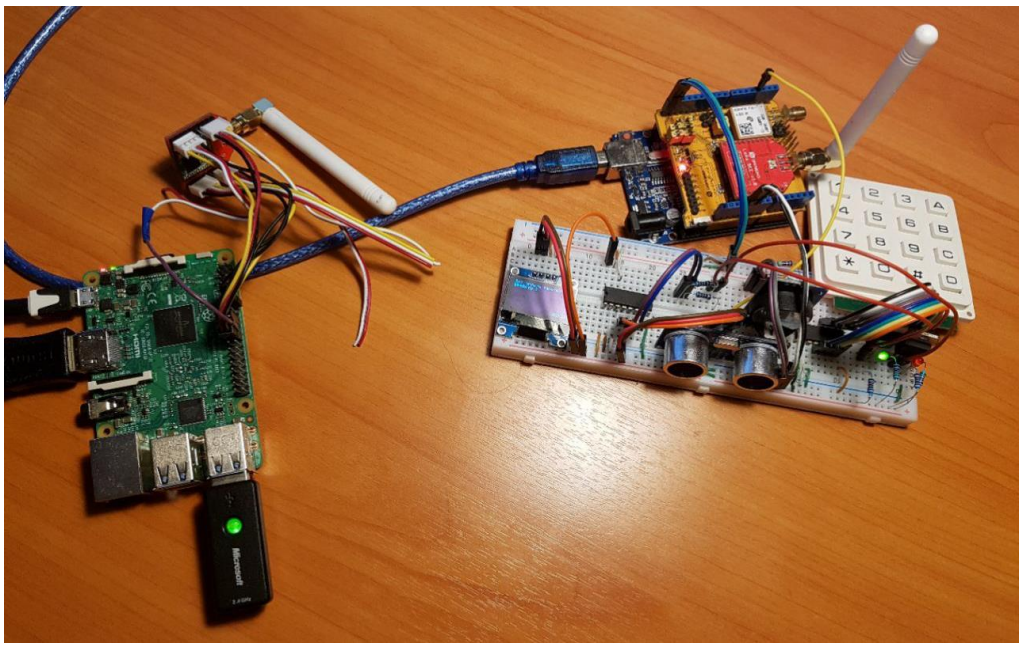

Рисунок В.1.1 – Внешний вид отладочного макета

Для создания макета использовались следующие элементы:

- Arduino Uno R3 – платформа для разработки на базе микроконтроллера ATMEGA328P, а также содержащая в себе драйвер USART-USB, выполненный на ATMEGA16U2.
- Макетная плата, которая используется для удобства подключения всех электронных компонентов.

- LoRa Dragino Shield – используется для удобства подключения модуля LoRa BEE к Arduino Uno R3.
- 2 радиомодуля Dragino LoRa BEE – первый для передачи / приёма данных на МКРСПМ, второй для передачи / приёма на вычислительном хабе Raspberry Pi. Данные модули сконструированы на базе модуля радиочастотного приёмопередатчика RFM95W.
- USB-кабель.
- 2 расширителя портов PCF8574P.
- Ультразвуковой датчик расстояния HC-SR04.
- Часы реального времени DS3231SN.
- Дисплей WAVGAT_128x64.
- Клавиатура AK-1604-N-WWB.
- Raspberry Pi 3 Model B, используемый в качестве вычислительного хаба.

B.1.2.2 Arduino Uno R3

Для отладки курсового проекта использовались реальные электронные компоненты. Микроконтроллер ATMEGA328 имеет отладочную плату, созданную на его основе - Arduino Uno R3, которая содержит драйвер USART-USB, выполненный на ATMEGA16U2, необходимый для передачи данных к ПЭВМ по последовательному интерфейсу. Внешний вид Arduino Uno R3 с обозначением входящих в него компонентов показан на рисунке B.1.2.

Рисунок B.1.2 – Внешний вид Arduino Uno R3 с обозначением входящих в него компонентов

Технические характеристики: [1]

- Микроконтроллер: ATMEGA328
- Рабочее напряжение: 5 В
- Входное напряжение (рекомендуемое): 7-12 В
- Входное напряжение (предельное): 6-20 В
- Цифровые Входы/Выходы: 14 (6 из которых могут использоваться как выходы ШИМ)
- Аналоговые входы: 6
- Постоянный ток через вход/выход: 40 мА
- Постоянный ток для вывода 3.3 В: 50 мА
- Флеш-память: 32 Кб (ATMEGA328) из которых 0.5 Кб используются для загрузчика
- ОЗУ: 2 Кб (ATMEGA328)
- EEPROM: 1 Кб (ATMEGA328)
- Тактовая частота: 16 МГц

Arduino Uno R3 получает питание через разъем USB, а встроенные в плату предохранители и стабилизаторы позволяют запитать микроконтроллер, драйвер и прочие элементы от стабильного напряжения 5В. Arduino Uno R3 также имеет специальный разъем для подачи напряжения от аккумулятора. Платформа может работать при внешнем питании от 6В до 20В. При напряжении питания ниже 7В вывод 5V может выдавать менее 5В, при этом платформа может работать нестабильно. При использовании напряжения выше 12В регулятор напряжения может перегреться и повредить плату. Рекомендуемый диапазон от 7В до 12В.

Плата имеет 14 цифровых входов / выходов (6 из которых могут использоваться как выходы ШИМ), силовой разъем, разъем ICSP и кнопку перезагрузки.

B.1.2.3 LoRa Dragino Sheild

Для удобства подключения модуля LoRa к микроконтроллеру используется шилд LoRa Dragino Shield. Расположение выводов данного шилда представлена на рисунке B.1.3.

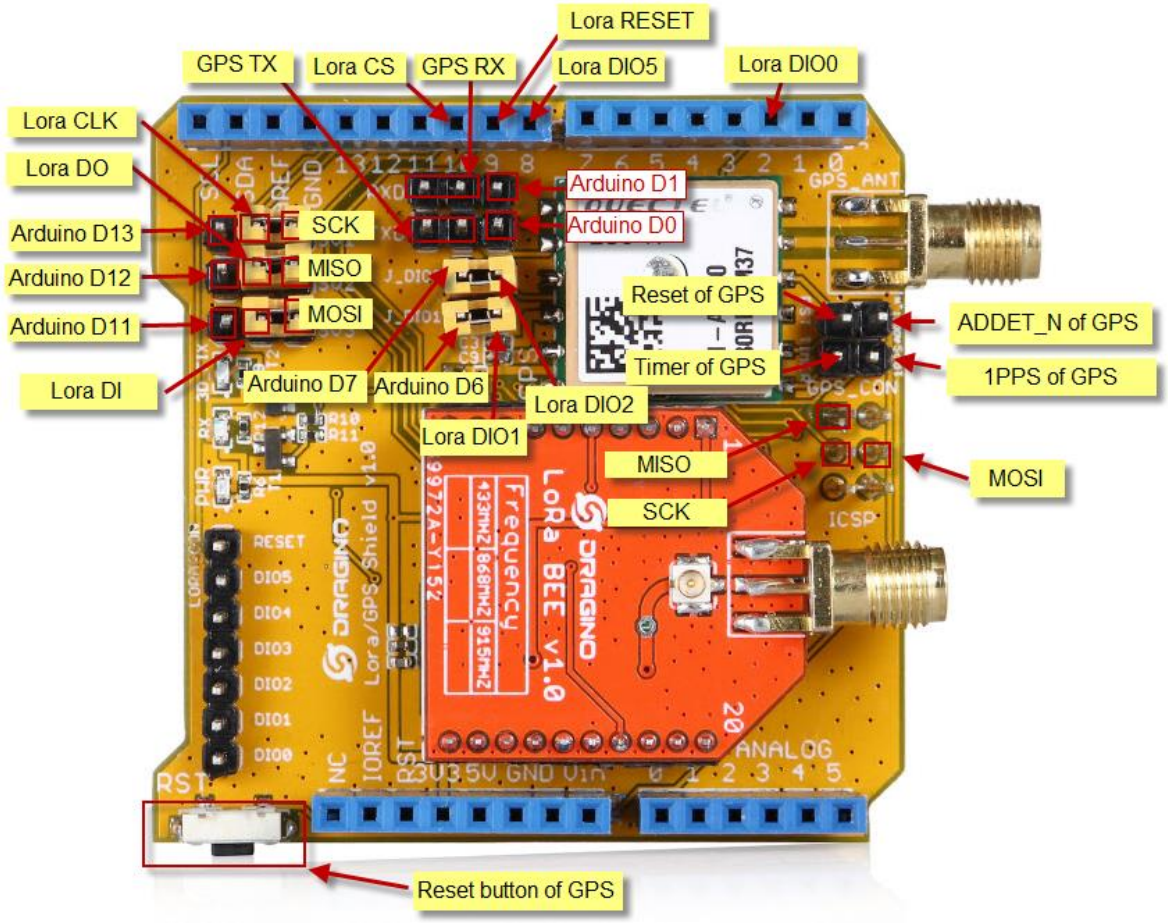

Рисунок B.1.3 – Расположение выводов LoRa Dagino Shield

На рисунке B.1.4 показано подключение данного шилда к микроконтроллеру Arduino UNO.

Рисунок B.1.4 – Внешний вид подключение LoRa Dagino Shield к Arduino UNO

Технические характеристики: [2]

- Диапазон частот: 868MHz/433MHz/915MHz;
- Низкое энергопотребление;
- Возможность соединения с Arduino UNO/Leonardo/Mega;
- Модем LoRa;
- FSK, GFSK, MSK, GMSK, LoRa/OOK модуляция;
- Определение преамбулы;
- Настройка пропускной способности;
- Встроенный датчик температуры;
- Индикатор заряда;
- GPS модуль автоматически переключает использование внутренней антенны и внешней.

B.1.2.4 Радиомодуль Dragino LoRa BEE

Dragino LoRa BEE - это LoRa-модуль, позволяющий отправлять данные на ультра-дальние расстояния с высокой помехоустойчивостью, обеспечивая при этом минимальное энергопотребление. Данный модуль основан на RFM95W передатчике, который поддерживает технологию модуляции LoRa и который широко применяется для задач, требующих больших расстояний для передачи данных. Модуль проявляет устойчивость к интерференционным помехам, что избавляет конечного разработчика от необходимости применения повторителей и создания сложной инфраструктуры сетей. Этот модуль может устанавливаться на посадочное место на плате расширения LoRa Dragino Shield. На рисунке B.1.5 показано расположение выводов данного модуля.

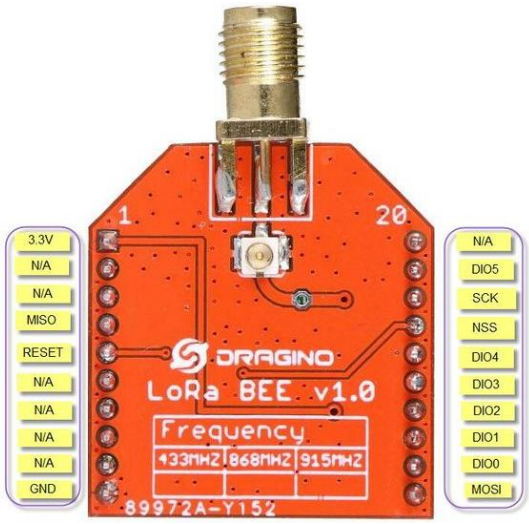

Рисунок B.1.5 – расположение выводов модуля Dragino Lora BEE

В.1.3 Тестирование отладочного макета

В.1.3.1 Тестирование случая отсутствия ТС на парковочном месте

Когда, парковочное место свободно, т.е. на пути сенсора нет объекта препятствия (т.е. ТС), зелёный светодиод должен гореть, что означает, что данное парковочное место свободно. Как можно видеть на рисунке В.1.6 на пути сенсора нет ни каких препятствий, и зелёный светодиод горит.

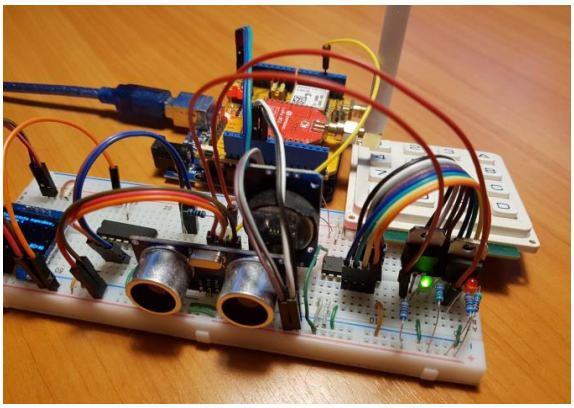

Рисунок В.1.6 – Тестирование случая отсутствия ТС на парковочном месте

В отладочном выводе (см. Рисунок В.1.7) видно, что было отправлено сообщение типа «Состояние парковочного места» (код типа 83) от МКПРСПМ с идентификатором 1 (2-е число), что парковочное место № 1 (3-е число) свободно (4-е число). А на рисунке В.1.8 показаны принимаемые сообщения вычислительным хабом Raspberry Pi (сообщения выводятся в 16-тиричном формате). Таким образом, данное тестирование пройдено успешно.

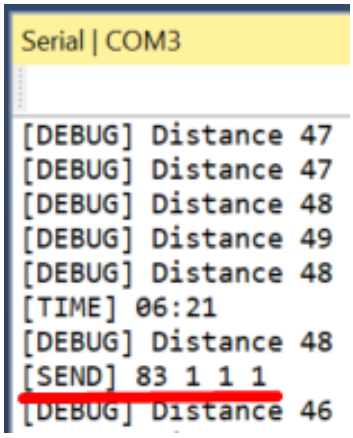

Рисунок В.1.7 – Отладочный вывод при отсутствии ТС на парковочном месте

Рисунок В.1.8 – Принимаемые пакеты Raspberry Pi при отсутствии ТС на парковочном месте

В.1.3.2 Тестирование случая присутствия ТС на парковочном месте

В данном случае. на пути сенсора будет стоять ТС. В этом случае зелёный светодиод должен быть погашен, что означает, что данное парковочное место занято. Как можно видеть на рисунке В.1.9 на пути сенсора находится автомобиль, и зелёный светодиод не горит.

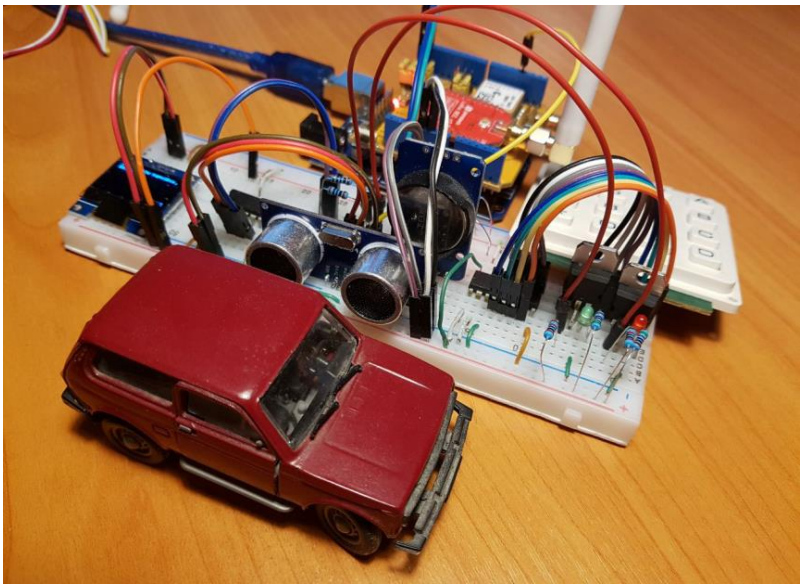

Рисунок В.1.9 – Тестирование случая присутствия ТС на парковочном месте

В отладочном выводе (см. Рисунок В.1.10) видно, что было отправлено сообщение типа «Состояние парковочного места» (код типа 83) от МКПРСПМ с идентификатором 1 (2-е число), что парковочное место № 1 (3-е число) занято (4-е число). На рисунке В.1.11 показаны принимаемые сообщения вычислительным хабом Raspberry Pi. Таким образом, данное тестирование пройдено успешно.

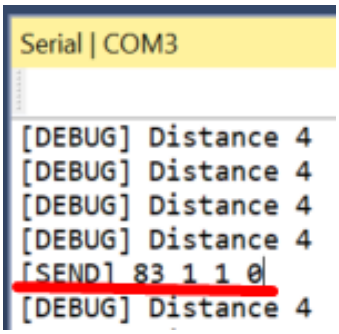

Рисунок В.1.10 – Отладочный вывод при присутствии ТС на парковочном месте

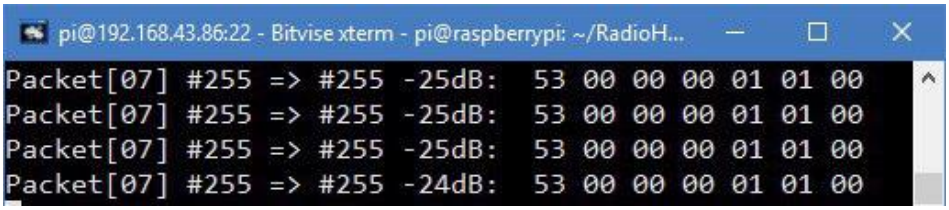

Рисунок В.1.11 – Принимаемые пакеты Raspberry Pi
при присутствии ТС на парковочном месте

В.1.3.3 Тестирование бронирования парковочного места

Красный светодиод горит, когда парковочное место забронировано, и погашен в ином случае. В начале парковочное место не забронировано, и на рисунке В.1.12 видно, что красный светодиод не горит.

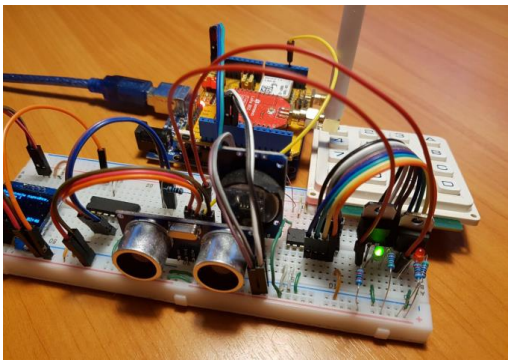

Рисунок В.1.12 – Парковочное место в состоянии «Не забронировано»

После, через USB интерфейс отправляется сообщение «\x0A\x72\x00\x00\x00\x01\x01\x00\x00\x00\x14» для бронирования парковочного места (см. Рисунок В.1.13). Структура этого сообщения:

- 1-й байт «\x0A» – длина сообщения (8 байт);
- 2-й байт «\x72» – тип сообщения («Забронировать парковочное место»);
- 3-й – 6-й байт «\x00\x00\x00\x01» – идентификатор МКПРСПМ (№ 1);
- 7-й байт «\x01» – номер парковочного места (№ 1);
- 8-й – 10-й байт «\x00\x00\x00\x14» – время бронирование (20 сек).

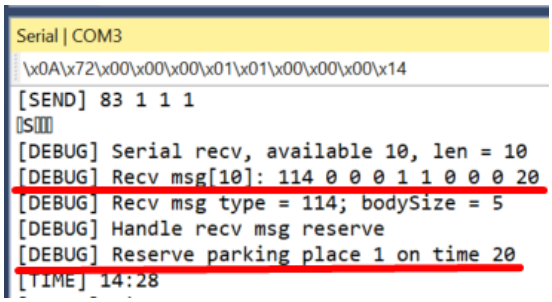

Рисунок В.1.13 - Отладочный вывод при отправки сообщения «Забронировать парковочное место»

Как видно в отладочном выводе (см. Рисунок В.1.13) микроконтроллер распознал сообщение и перевел указанное парковочное место на 20 сек в состояние забронировано. На рисунке В.1.14 показано, что красный светодиод загорелся, что означает, что данное парковочное место находится в состоянии «Забронировано».

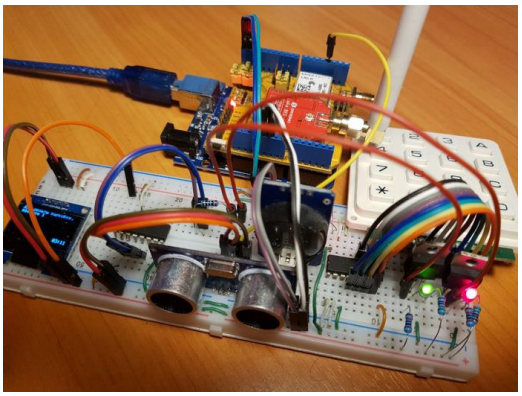

Рисунок В.1.14 – Парковочное место переведено в состояние «Забронировано»

Через 20 сек светодиод снова погас (см. Рисунок В.1.15), т.е. парковочное место обратно переведено в состояние «Не забронировано». Таким образом, данное тестирование пройдено успешно.

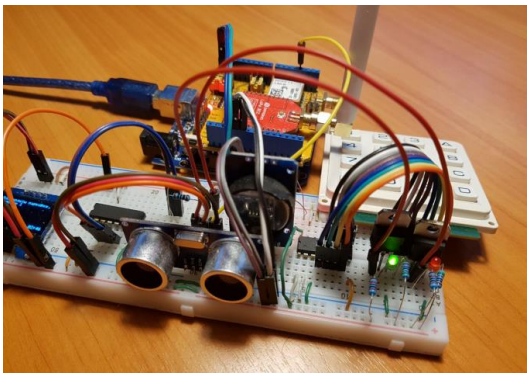

Рисунок В.1.15 – Парковочное место обратно переведено в состояние «Не забронировано»

В.1.3.4 Тестирование оплаты парковочного места

В начальный момент времени на дисплее отображается «Начальный экран» (см. Рисунок В.1.16)

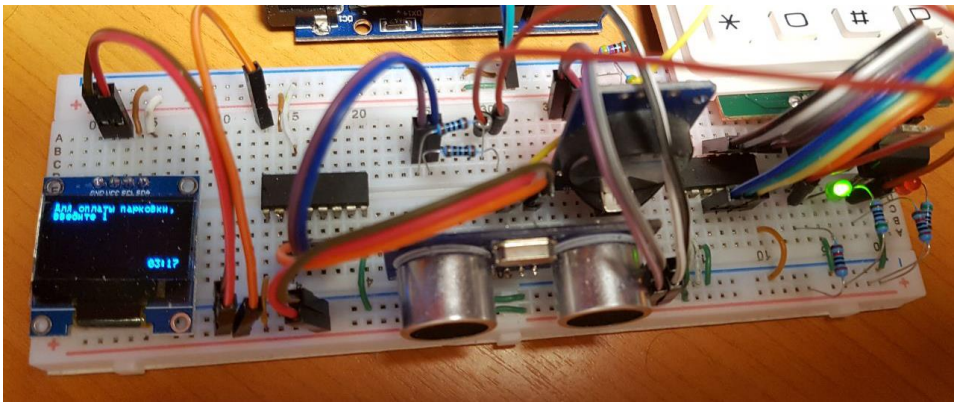

Рисунок В.1.16 – «Начальный экран» на дисплее

Далее пользователь вводит 1 на клавиатуре, и состояние терминала оплаты изменяется на «Ввод номера парковки» (см. Рисунок В.1.17).

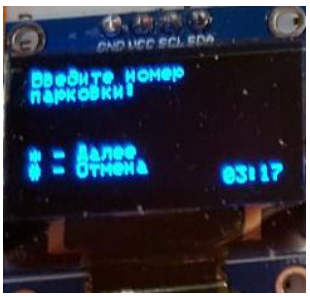

Рисунок В.1.17 – Начало ввода номера парковочного места

После чего пользователь начинает ввод номера парковки, и на дисплее сразу отображается введение число (см. Рисунок В.1.18).

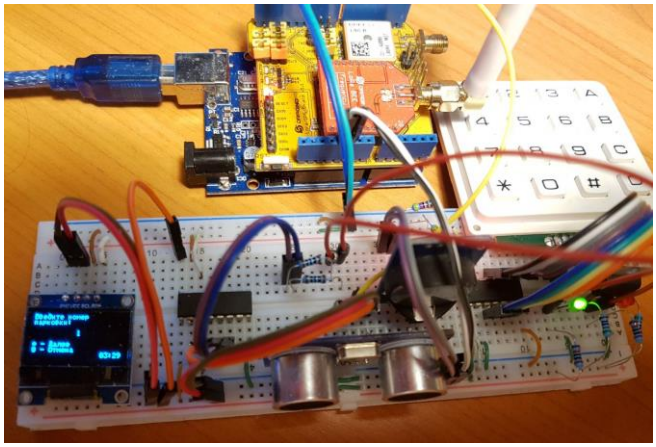

Рисунок В.1.18 - Ввод номера парковочного места

После ввода, пользователь должен нажать «*» для продолжения или «#» для отмены. Если пользователь нажал «*», то терминал оплаты переводится в состояние «Ввод времени», и пользователь начинает ввод.

Рисунок В.1.19 – Ввод времени

Когда пользователь введет время и нажмет «*», микроконтроллер подсчитает стоимость к оплате и выдаст эту сумму к оплате. Пользователю нужно будет ввезти сумму для оплаты (см. Рисунок В.1.20).

Рисунок В.1.20 – Оплата

После «ввода средств» микроконтроллер проверит, что введённая сумма больше суммы к оплате, и выведет сообщение «Оплата произведена», выведет сдачу и переведет состояние парковочного места в «Забронировано». Следовательно красный светодиод должен зажечься, как и произошло (см. Рисунок В.1.21).

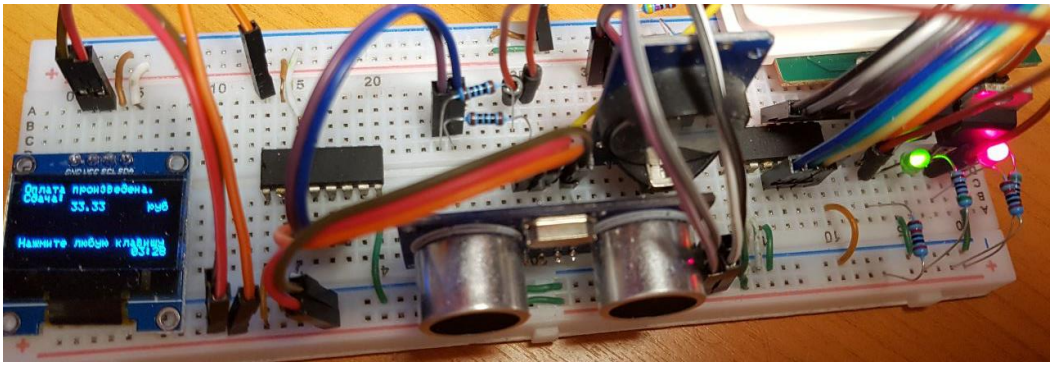

Рисунок В.1.21 – Оплата произведена

В случае если произошла какая-либо ошибка (недостаточно средств, данные не введены, таймаут, платёж отменен), то выводится соответствующее сообщение об ошибке (см. Рисунок В.1.22). Таким образом, данное тестирование пройдено успешно.

Рисунок В.1.22 – Сообщения об ошибках

В.1.3.5 Результаты тестирования ПО МКПРСПМ

В таблице 1 приведены результаты проведённых тестирований. Таким образом, согласно результатам тестирований собранного макета выяснилось, что разработанная МК-подсистема функционирует без ошибок и успешно выполняет все функции.

Таблица 1 – Результаты тестирований

Наименование тестирования	Результат
Тестирование случая отсутствия ТС на парковочном месте	Пройдено
Тестирование случая присутствия ТС на парковочном месте	Пройдено
Тестирование бронирования парковочного места	Пройдено
Тестирование оплаты парковочного места	Пройдено

В.2 Тестирование вычислительного хаба

Ниже в таблице 2 приводится результаты проведенного тестирования вычислительного хаба принятия данных от датчика МКПРСПМ:

Таблица 2 – Тестирование принятия данных от датчика МКПРСПМ

Номер теста	Назначение теста	Введенные данные	Реакция программы	Вывод
1	Подключение к датчику по UART и принятие сообщения о состоянии ПМ	Передача по UART: `\x07\x53\x00\x00 \x00\x01\x01\x01`	Сообщение было принято от датчика №1. Парковочное место №1 свободно. Отправка по MQTT: `{"login":"user@mail.ru", "password": "pwd123", "sensor": 1, "place_id": 1, "free", true}`	Тест пройден
2	Подключение к датчику по LoRa и принятие сообщения о состоянии ПМ	Передача по LoRa: `\x53\x00\x00\x00 \x01\x01\x01`	Сообщение было принято от датчика №1. ПМ №1 свободно. Отправка по MQTT: `{"login":"user@mail.ru", "password": "pwd123", "sensor": 1, "place_id": 1, "free", true}`	Тест пройден
3	Принятие сообщения от датчика об инициализации	`\x49\x00\x00\x00 \x01\x00\xFF\x10 \x00\x00\xFF\x00 \xFF\x00\x00\x00 \xFF\x00\x00\x00 \xFF`	Сообщение было принято от датчика №1. Отправка по MQTT: `{"login":"user@mail.ru", "password": "pwd123", "sensor": 1, "sampling_period": 255, "sending_period": 4096, "day_cost": 255, "night_cost": 255, "day_start_time": 255, "night_start_time": 255}`	Тест пройден
4	Принятие сообщения от датчика об оплате бронирования парковочного места	`\x50\x00\x00\x00 \x01\x01\x00\x00 \x00\xFF\x00\xFF \x00\xFF`	Сообщение было принято от датчика №1. Оплата ПМ №1 Отправка по MQTT: `{"login":"user@mail.ru", "password": "pwd123", "sensor": 1, "place_id": 1, "booked_time": 255, "payment": 255, "total_cost": 255}`	Тест пройден

Для тестирования принятия данный через MQTT использовался MQTT-клиент «mosquitto-clients». Так, например, чтобы опубликовать данные на MQTT-брокере используется команда «mosquitto_pub». Ниже в таблице 3 приводится результаты проведенного тестирования вычислительного хаба принятия команд от сервера принятых по MQTT:

Таблица 3 – Тестирование принятия команд от сервера

Номер теста	Назначение теста	Введенные данные	Реакция программы	Вывод
1	Принятие команды бронирования ПМ	`$ mosquitto_pub -d -t sensor_1-book -m '{\ "place_id": 1,\ "booking_time": 255}'`	Команда для датчика №1. Бронирование ПМ №1 на 255 сек. Отправка команды: `\x72\x00\x00\x00 \x01\x01\x00\x00 \x00\xFF`	Тест пройден
2	Принятие команды изменения параметров датчика	`$ mosquitto_pub -d -t sensor_1-book -m '{\ "sampling_period": 255,\ "sending_period": 4096,\ "day_cost": 255,\ "night_cost": 255,\ "day_start_time": 255,\ "night_start_time": 255}'`	Команда для датчика №1. Изменить параметры. Отправка команды: `\x73\x00\x00\x00 \x01\x00\xFF\x10 \x00\x00\xFF\x00 \xFF\x00\x00\x00 \xFF\x00\x00\x00 \xFF`	Тест пройден

Ниже в листинге 1 показан пример отладочного вывода ПО вычислительного хаба:

Листинг 1 – Отладочный вывод ПО вычислительного хаба

```
pi@raspberrypi:~/SmartParking-Transceiver $ sudo ./SmartParking-Transceiver
MQTT connect to  "0.0.0.0"
MQTT connect to  "192.168.43.139"
MQTT connect to  "m23.cloudmqtt.com"
MQTT connected to: "m23.cloudmqtt.com"
Subscribe to  "sensor_1-book"
Subscribe to  "sensor_1-settings"
Subscribe to  "sensor_2-book"
Subscribe to  "sensor_2-settings"
Recieve message
[RECV] "S\x00\x00\x00\x01\x01\x01"
[DEBUG] Recv msg type: 83 ; sensor ID: 1
[PARKING STATUS] ID = 1 , place = 1 , is free = true
MQTT connect to  "0.0.0.0"
MQTT connect to  "192.168.43.139"
Recieve message
[RECV] "S\x00\x00\x00\x01\x01\x01"
[DEBUG] Recv msg type: 83 ; sensor ID: 1
[PARKING STATUS] ID = 1 , place = 1 , is free = true
MQTT connect to  "0.0.0.0"
MQTT connect to  "192.168.43.139"
```

В.3 Тестирование сервера

В.3.1 Стратегии тестирования

В составе Ruby on Rails есть отличные средства автоматизированного тестирования. Rails предлагает писать тесты очень просто – при создании модели и контроллера, он начинает создавать скелет тестового кода. Простой запуск тестов Rails позволяет убедиться, что код придерживается нужной функциональности даже после большой переделки кода.

В Rails тестирование различных экшнов контроллера — это форма написания функциональных тестов. При написании функциональных тестов, тестируется, как экшны обрабатывают запросы, ожидаемый результат и отклики представлений HTML.

После проверки на наличие ошибок приступают к оценочному тестированию - проверка на соответствие нормам и стандартам. Данный метод тестирования позволяет оценить окончательную готовность программного продукта.

В.3.2 Автоматизированное тестирование

Для разработанного веб-приложения были написаны 60 автоматизированных теста, покрывающие весь функционал. Из них 14 модульных теста, 6 интеграционных теста и 40 функциональных тестов контроллеров. Для запуска тестирования требуется вбить команду «rails test». Ниже в листинге 2 приводится вывод тестирования, в котором говорится, что все 60 тестов успешно пройдены.

Листинг 2 – Результаты автоматизированного тестирования веб-приложения

```
$ rails test
Running via Spring preloader in process 19776
Run options: --seed 62527

# Running:

............................................................

Finished in 1.669284s, 35.9436 runs/s, 60.5050 assertions/s.
60 runs, 101 assertions, 0 failures, 0 errors, 0 skips
```

В.3.3 Нагрузочное тестирование

Основной задачей любого тестирования производительности сайта является понимание его устойчивости к нагрузкам, которые могут появляться не только из-за большого количества

посетителей онлайн, но и являться следствием некорректной настройки сервера, неправильной работы скриптов или действиями злоумышленников (DOS, DDOS). Было проведено нагрузочное тестирование при помощи утилиты Apache Benchmark.

Технические характеристики тестируемого оборудования:

- Процессор: Intel Core I5-6600
- ОЗУ: 8 Гб

Ниже в листингах 3, 4 приведены результаты нагрузочного тестирования для выдачи главной страницы сайта и раздачи статики.

Листинг 3 – Результаты нагрузочного тестирования выдачи главной страницы

```
$ ab -n 10000 -c 100 http://0.0.0.0/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>

Benchmarking 0.0.0.0 (be patient)
Finished 10000 requests

Server Software:        nginx
Server Hostname:        0.0.0.0
Server Port:            80

Document Path:          /
Document Length:        3086 bytes
Concurrency Level:      100
Time taken for tests:   12.266 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      38674350 bytes
HTML transferred:       30860000 bytes
Requests per second:    815.28 [#/sec] (mean)
Time per request:       122.658 [ms] (mean)
Time per request:       1.227 [ms] (mean, across all concurrent requests)
Transfer rate:          3079.13 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       4
Processing:     5  121  66.1    108     396
Waiting:        5  121  66.1    108     396
Total:          5  121  66.1    108     396

Percentage of the requests served within a certain time (ms)
  50%    108
  66%    138
  75%    160
  80%    174
  90%    213
  95%    248
  98%    292
  99%    321
 100%    396 (longest request)
```

Листинг 4 – Результаты нагрузочного тестирования выдачи статики

```
$ ab -n 10000 -c 100 http://0.0.0.0/assets/application-
92b4608058c149877b3d09f90933d850ebf558134609a4b24b2e0d2d90caccba.js
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>

Benchmarking 0.0.0.0 (be patient)
Finished 10000 requests

Server Software:        nginx
Server Hostname:        0.0.0.0
Server Port:            80

Document Path:          /assets/application-
92b4608058c149877b3d09f90933d850ebf558134609a4b24b2e0d2d90caccba.js
Document Length:        259697 bytes

Concurrency Level:      100
Time taken for tests:   1.704 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      2600380000 bytes
HTML transferred:       2596970000 bytes
Requests per second:    5867.61 [#/sec] (mean)
Time per request:       17.043 [ms] (mean)
Time per request:       0.170 [ms] (mean, across all concurrent requests)
Transfer rate:          1490040.08 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.6      0       8
Processing:     2   17   5.7     16      51
Waiting:        0   15   5.0     15      49
Total:          2   17   5.9     16      58

Percentage of the requests served within a certain time (ms)
  50%     16
  66%     18
  75%     19
  80%     20
  90%     23
  95%     27
  98%     32
  99%     41
 100%     58 (longest request)
```

Как можно видеть из результатов тестирования (см. листинги 3, 4) количество запросов в секунду на выдачу главной страницы составило 815 запросов в секунду и для статики – 5867 запросов в секунду.

Ниже в листинге 5 приводятся результаты нагрузочного тестирования алгоритма выдачи данных о состоянии парковочных мест. При тестировании БД была заполнена: в таблице «Стоянки» находилось 536 записей, в таблице «Парковочные места» – 13046 записей.

Листинг 5 – Результаты нагрузочного тестирования алгоритма выдачи парковок

```
$ ab -n 10000 -c 100
'http://0.0.0.0:3000/api/parkings?radius=0.02&coord[lat]=55.63&coord[lng]=37.52'
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>

Benchmarking 0.0.0.0 (be patient)
Finished 10000 requests

Server Software:
Server Hostname:        0.0.0.0
Server Port:            3000

Document Path:
/api/parkings?radius=0.02&coord[lat]=55.63&coord[lng]=37.52
Document Length:        6510 bytes

Concurrency Level:      100
Time taken for tests:   11.810 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      69590000 bytes
HTML transferred:       65100000 bytes
Requests per second:    846.74 [#/sec] (mean)
Time per request:       118.099 [ms] (mean)
Time per request:       1.181 [ms] (mean, across all concurrent requests)
Transfer rate:          5754.39 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.3      0      18
Processing:     4  115  71.4     98     437
Waiting:        4  115  71.4     98     437
Total:          4  115  71.7     98     437

Percentage of the requests served within a certain time (ms)
  50%     98
  66%    130
  75%    154
  80%    170
  90%    219
  95%    255
  98%    309
  99%    339
 100%    437 (longest request)
```

Таким образом, количество запросов в секунду для алгоритма выдачи парковочных мест для заполненной БД с общим объёмом 13582 записей составило 846 запросов в секунду.

В.4 Комплексное тестирование системы

В.4.1 Тестирование изменения состояния парковочного места

Изначально пусть на пути сенсора не будет преград, т.е. парковочное место свободно. Горящий зеленый светодиод у датчика МКПРСПМ (см. Рисунок В.4.1) подтверждает, что машиноместо свободно.

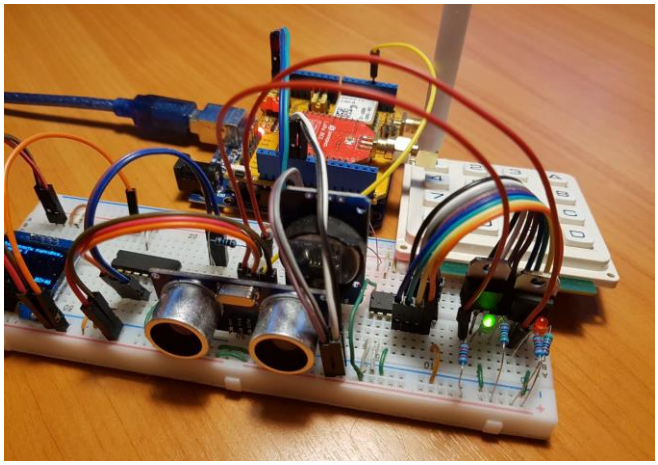

Рисунок В.4.1 – Тестирование случая отсутствия ТС на парковочном месте

Датчик отправил по сети LoRaWAN данные, что парковочное место свободно, и вычислительный хаб принимает данное сообщение, что подтверждает вывод в листинге 6:

Листинг 6 – Вывод ПО вычислительного хаба о принятия сообщения, что ПМ свободно

```
[RECV] "S\x00\x00\x00\x01\x01\x01"
[DEBUG] Recv msg type: 83 ; sensor ID: 1
[PARKING STATUS] ID = 1 , place = 1 , is free = true
```

Вычислительный хаб отправил это сообщение на сервер. Теперь данное парковочное место отображается на карте, как свободное (см. Рисунок В.4.2). Таким образом данная проверка пройдена.

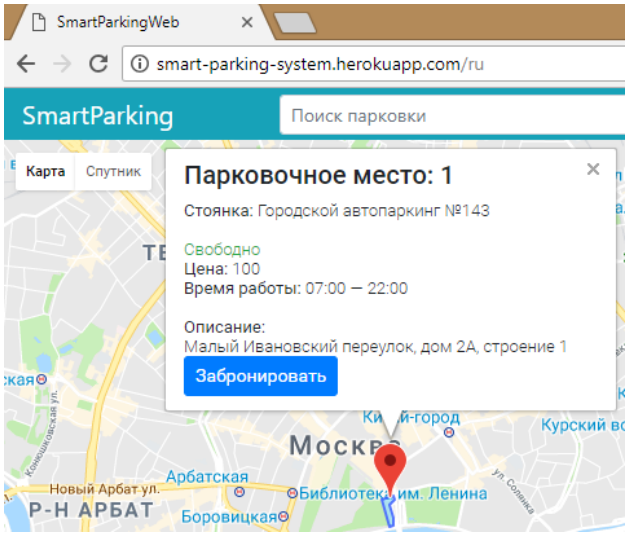

Рисунок В.4.2 – Парковочное место отображается как свободное

Теперь пусть на пути сенсоры будет преграда, таким образом ПМ будет занято. У датчика МКПРСПМ зеленый светодиод погас.

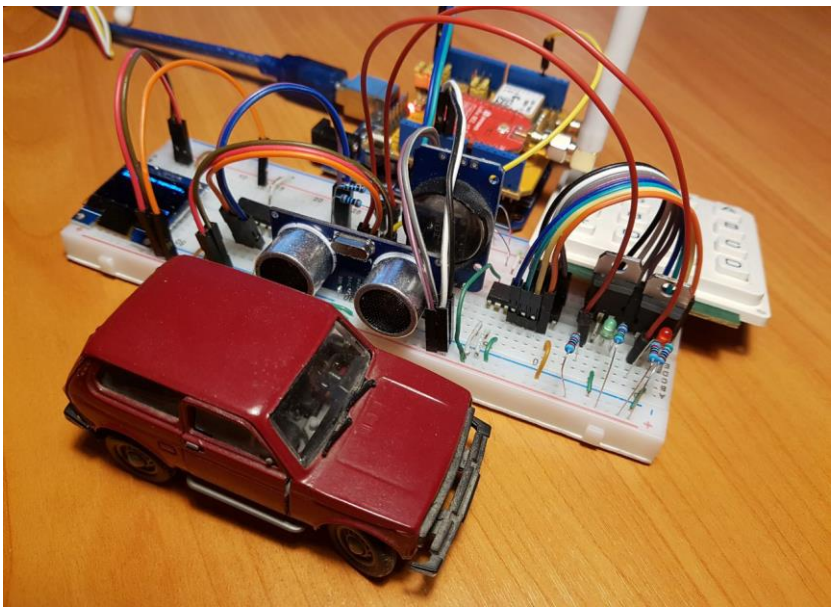

Рисунок В.4.3 – Тестирование случая присутствия ТС на парковочном месте

Датчик отправил по сети LoRaWAN сообщение, что парковочное место занято. Вычислительный хаб принимает данное сообщение, что подтверждает вывод в листинге 7:

Листинг 7 – Вывод ПО вычислительного хаба о принятия сообщения, что ПМ занято

```
[RECV] "S\x00\x00\x00\x01\x01\x00"
[DEBUG] Recv msg type: 83 ; sensor ID: 1
[PARKING STATUS] ID = 1 , place = 1 , is free = false
```

Вычислительный хаб отправил это сообщение на сервер. Теперь данное парковочное место отображается на карте, как занятое (см. Рисунок В.4.4). Таким образом тестирование успешно пройдено.

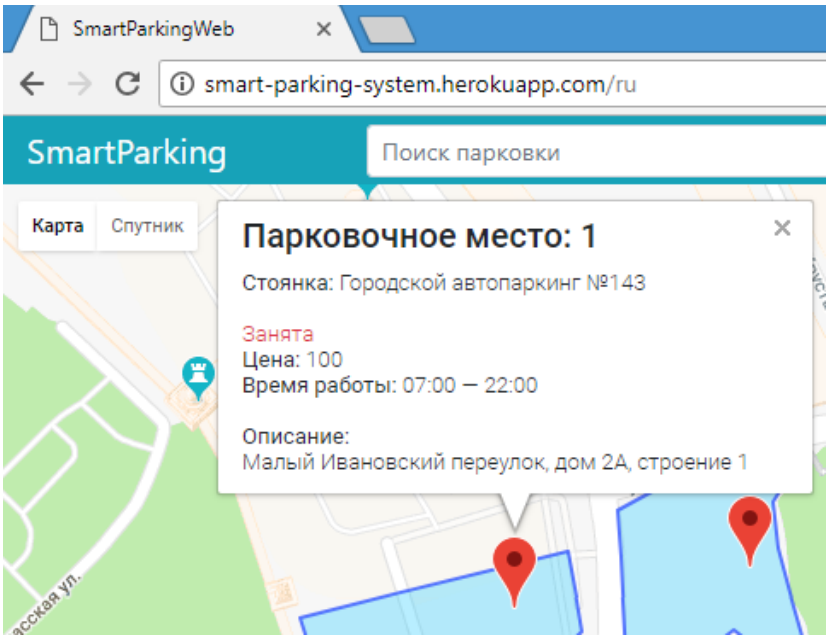

Рисунок В.4.4 – Парковочное место отображается на карте как занятое

B.4.2 Тестирование онлайн бронирования

Если парковочное место в данный момент свободно, то имеется возможность онлайн бронирования его. Забронируем его на 1 минуту (см. Рисунок B.4.5). Теперь ПМ на карте отображается как забронированное (см. Рисунок B.4.6).

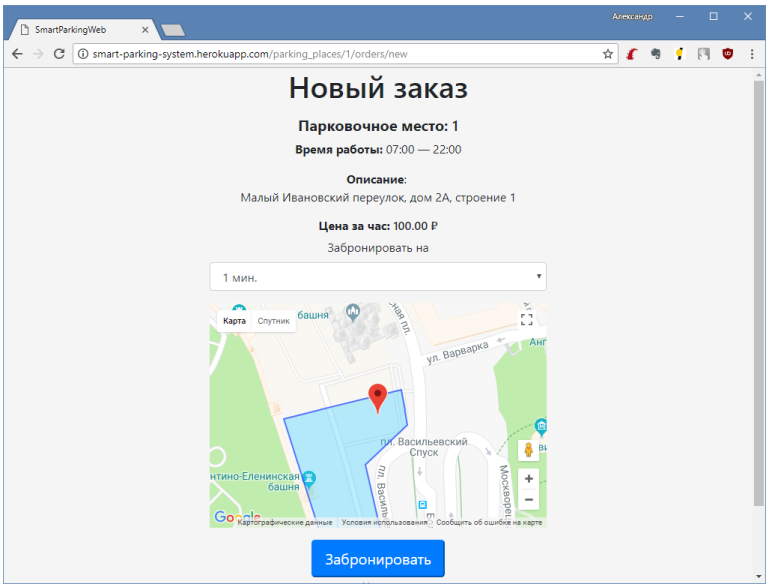

Рисунок B.4.5 – Онлайн бронирование парковочного места

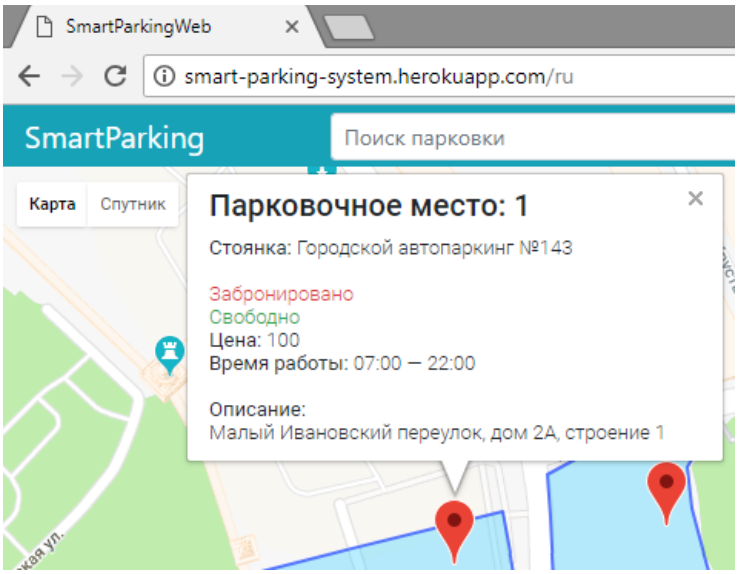

Рисунок B.4.6 – Парковочное место забронировано

Сервер отправил MQTT-сообщение для бронирования парковочного места. Вычислительный хаб его принял (см. Листинг 8) и отправил команду датчику по сети LoRaWAN.

Листинг 8 – Вывод ПО вычислительного хаба о принятии команды бронирования ПМ

```
[RECV FROM SERVER] topic: "sensor_1-book" ; msg:
"{\"place_id\":1,\"booking_time\":60}"
[BOOK] ID = 1 , place ID = 1 , booked time = 60
[SEND] Msg[ 10 ]: "r\x00\x00\x00\x01\x01\x00\x00\x00<"
```

Датчик МКПРСПМ принял команду и зарезервировал данное парковочное место и зажегся красный светодиод (см. Рисунок В.4.7) на 1 минуту. Таким образом тестирование успешно пройдено.

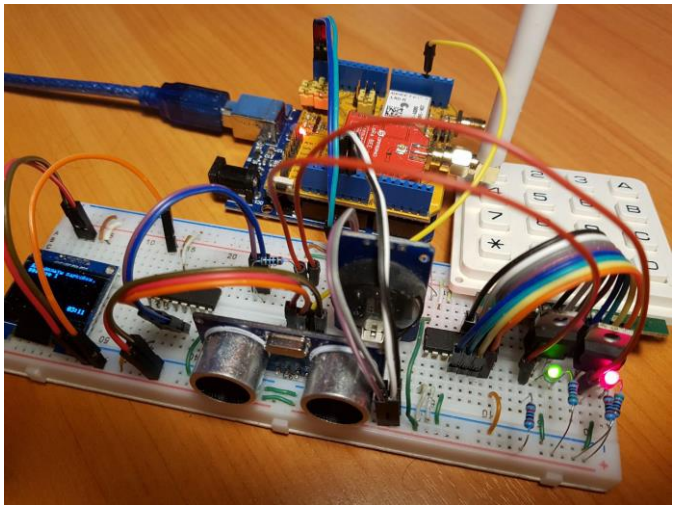

Рисунок В.4.7 – Парковочное место переведено в состояние «Забронировано»

В.4.3 Тестирование бронирования через терминал оплаты

Через терминал оплаты забронируем парковочное место на 1 минуту (см. Рисунок В.4.8). Загорелся красный светодиод, что означает, ПМ забронировано.

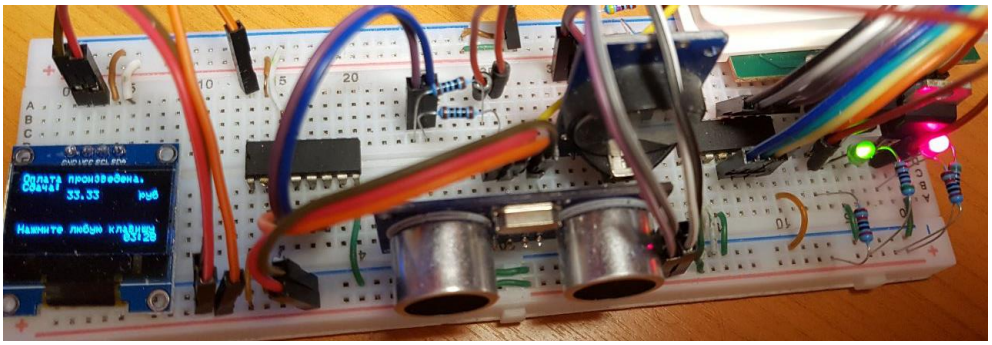

Рисунок В.4.8 – Бронирование через терминал оплаты

МКПРСПМ отправило сообщение о бронировании парковочного места. Вычислительный хаб принял данное сообщения (см. Листинг 9) и отправил его серверу.

Листинг 9 – Вывод ПО вычислительного хаба о принятии сообщения об оплате

```
[RECV] "P\x00\x00\x00\x01\x01\x00\x00\x00<\x00\x05\x00\x01"
[DEBUG] Recv msg type: 80 ; sensor ID: 1
[PAYMENT TERMINAL] ID = 1 , place ID = 1 , booked time = 60 , payment = 5 ,
total cost = 1
```

Теперь в БД добавился еще 1 заказ от терминала оплаты (см. Рисунок В.4.9), а также данное проковочное место на карте отображается как забронированное (см. Рисунок В.4.10). Таким образом тестирование успешно пройдено.

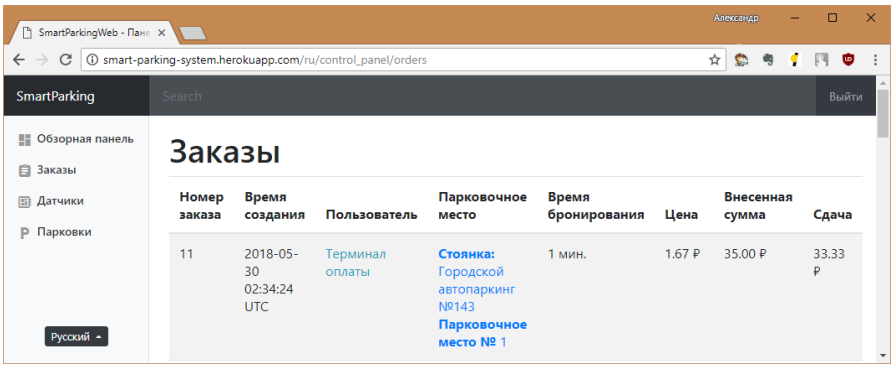

Рисунок В.4.9 – Добавился заказ от терминала оплаты

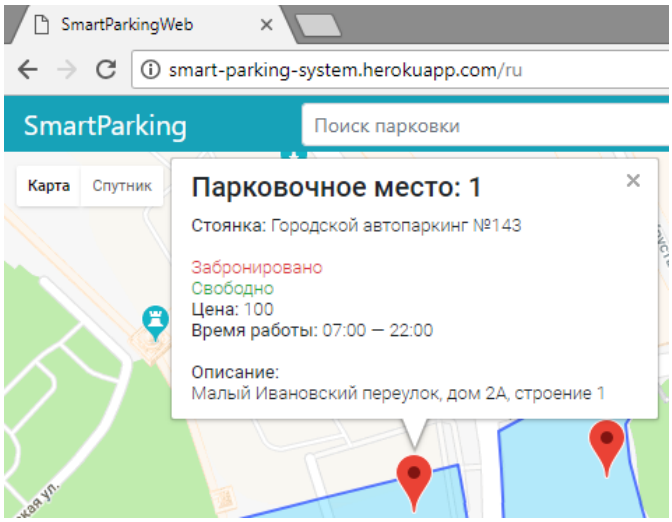

Рисунок В.4.10 – Парковочное место забронировано

В.4.4 Тестирование удаленной настройки датчика

На странице изменения параметров датчика «*/sensors/1/edit*» изменим время отправки данных на 5000 мс и стоимость на 100 руб (см. Рисунок В.4.11).

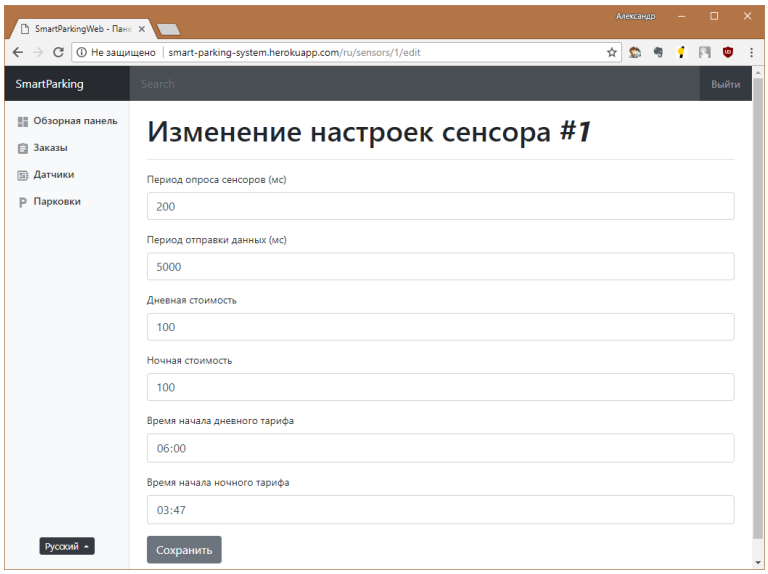

Рисунок В.4.11 – Изменение настроек датчика

Сервер отправил команду на изменения настроек. Вычислительный хаб её принял и отправил датчику (см. Листинг 10).

Листинг 10 – Вывод ПО вычислительного хаба о принятии команды изменения настроек

```
[RECV FROM SERVER] topic: "sensor_1-settings" ; msg:
"{\"sampling_period\":200,\"sending_period\":5000,\"day_cost\":100,\"night_cost\
":100,\"day_start_time\":21600,\"night_start_time\":13620}"
[SET SENSOR SETTINGS] ID = 1 , sampling period = 200 , sending period = 5000 ,
day cost = 100 , night cost = 100 , day start time = 21600 , night start time =
13620
[SEND] Msg[ 21 ]:
"s\x00\x00\x00\x01\x00\xC8\x13\x88\x00""d\x00""d\x00\x00T`\x00\x00""54"
Recieve message
[RECV] "I\x00\x00\x00\x01\x00\xC8\x13\x88\x00""d\x00""d\x00\x00T`\x00\x00""54"
[DEBUG] Recv msg type: 73 ; sensor ID: 1
[INIT] ID = 1 , sampling period = 200 , sending period = 5000 , day cost = 100 ,
night cost = 100 , day start time = 21600 , night start time = 13620
```

МКПРСПМ принял команду, изменил параметры и отправил сообщение об инициализации. И вычислительный хаб принял сообщение об инициализации и отправил серверу (см Листинг 10). Теперь данные от МКПРСПМ отправляются каждые 5 секунд, а стоимость парковки составляет 100 руб (см. Рисунок В.4.12)

Рисунок В.4.12 – Стоимость за 1 час брони парковочного места через терминал оплаты

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Общие сведения об ArduinoUno [Электронный ресурс]. – URL: http://arduino.ru/Hardware/ArduinoBoardUno (дата обращения 10.11.2017).

2 Руководство для LoRa Dragino Shield [Электронный ресурс]. – http://wiki.dragino.com/index.php?title=Lora_Shield.