UniCEUB - Centro Universitário de Brasília
Faculdade de Tecnologia e Ciências Sociais Aplicadas

ALLAN ALVES

SISTEMA DE SEGURANÇA VEICULAR COM MONITORAÇÃO POR SMARTPHONE

Curso de Engenharia de Computação
Brasília - Distrito Federal, 2016

ALLAN ALVES

SISTEMA DE SEGURANÇA VEICULAR COM MONITORAÇÃO POR SMARTPHONE

Projeto de finalização do curso de Engenharia de Computação da Faculdade de Tecnologia e Ciências Sociais Aplicadas (FATECS) do Centro Universitário de Brasília (UniCEUB). Desenvolvido sob orientação do Prof. Francisco Javier de Obaldía Díaz e apresentado na instituição como requisito para obtenção do título de Bacharel.

UniCEUB - Centro Universitário de Brasília
Faculdade de Tecnologia e Ciências Sociais Aplicadas
Curso de Engenharia de Computação
Brasília - Distrito Federal, 2016

ALLAN ALVES

SISTEMA DE SEGURANÇA VEICULAR COM MONITORAÇÃO POR SMARTPHONE

Projeto de finalização do curso de Engenharia de Computação da Faculdade de Tecnologia e Ciências Sociais Aplicadas (FATECS) do Centro Universitário de Brasília (UniCEUB). Desenvolvido sob orientação do Prof. Francisco Javier de Obaldía Díaz e apresentado na instituição como requisito para obtenção do título de Bacharel.

Este trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Dr. Abiezer Amarilia Fernandes

Coordenador do Curso

Banca examinadora:

_____

Prof. Francisco Javier de Obaldía Díaz, Mestrado.

Orientador

_____

_____

_____

Agradecimentos

Agradeço primeiramente à minha família por me apoiar e ajudar a me manter determinado nos longos dias de trabalho.

Ao professor Orientador Francisco Javier, pelo apoio e dedicação no desenvolvimento do projeto, e pelo empenho demonstrado em ajudar seus orientados.

Aos professores da instituição, que se importam com o aprendizado de seus alunos e se preocupam em passar o máximo de conhecimento e se motivarem com a vontade de aprender de seus alunos.

Ao meu amigo Luiz Eduardo Sol, pelos momentos de ajuda e por passar seu valioso conhecimento de futuro Engenheiro.

À minha namorada Rayza Adrielle por me mostrar a importância da dedicação aos estudos para alcançar meus objetivos de vida, e por ajudar a me manter motivado ao longo de todo o trabalho.

Por fim, à todos que contribuíram, direta ou indiretamente, para que concluísse este projeto.

Sumário

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Lista de Símbolos e Siglas

ANSI American National Standards Institute

API Application Programming Interface

API Application Programming Interface

APNs Apple Push Notification Service

AT Conjunto de comandos Hayes, conhecidos como ’Comandos AT’

FTDI Future Technology Devices International

GND Ground

GPRS General Packet Radio Service

GSM Global System for Mobile Communications

HTTP Hypertext Transfer Protocol

ICC-ID Integrated Circuit Card Identifier

IMSI International Mobile Subscriber Identity

JSON JavaScript Object Notation

MAC Media Access Control

MBaaS Mobile Backend as a Service

PIR Passive Infrared Sensor

PWM Pulse Width Modulation

ROM Read-Only Memory

RS Recommended Standard

RX Recepção

SDK Software Development Kit

SIM Subscriber Identity Module

SMA SubMiniature version A

SMS Short Message Service

SSID Service Set Identifier

TCP/IP Transmission Control Protocol

TTL Transistor-Transistor Logic

TX Transmissão

UART Universal asynchronous receiver/transmitter

URA Unidade de Resposta Audível

USB Universal Serial Bus

VCC Integrated Circuits power-supply pin

Resumo

Pelos altos índices de crimes como furto e roubo de automóveis, novas tecnologias tendem a ser desenvolvidas. Com a popularidade do smartphone e a importância de seus aplicativos, nos últimos anos seu uso tornou-se praticamente indispensável na vida das pessoas. A monitoração de um veículo pessoal pode se tornar uma das principais funções dos aplicativos, sendo utilizados para ajudar a evitar possíveis reações por parte de donos de veículos em eventuais roubos informando a localização do dispositivo instalado internamente. Informar eventos como suspeita de invasão e acesso ao veículo, bem como facilitar seu acesso utilizando seu smartphone para permissão, também são funcionalidades que podem manter o usuário ciente da situação do seu veículo pessoal a qualquer momento. Com o intuito de se desenvolver um sistema de segurança com interação por aplicativo, tendo um dispositivo que pode ser facilmente instalado, desenvolveu-se este projeto. Este trabalho, além da implementação do dispositivo físico, inclui a criação de um aplicativo na plataforma iOS e um serviço em nuvem para consumo pelo aplicativo e pelo dispositivo instalado no veículo. Com a integração destas partes, foi possível a construção de um sistema que informa o usuário sobre eventos físicos e localização. Instabilidades que foram detectadas em testes variados, foram resolvidas utilizando-se alimentação com corrente suficiente para todos os módulos e com o uso de capacitores. Estas instabilidades foram notadas principalmente no funcionamento do módulo WiFi, ESP8266, que mostrou-se uma placa bastante sensível. O uso do serviço MBaaS permitiu a implementação de uma série de funcionalidades com pouco esforço, resumindo as necessidades do sistema em simples interações na aplicação do serviço e implementação de toda sua lógica em apenas um arquivo em JavaScript para a interação com o aplicativo iOS e o dispositivo físico. Notou-se que a qualidade e o tipo do sensor de presença é um importante fator do sistema, necessitando de técnicas mais estudadas para garantir que não haja emissões falsas de suspeita de intrusão.

Palavras Chave: Smartphone, iOS, GSM, WiFi, MBaaS, Nuvem, Arduino, Sensor PIR, APNS, Objective-C, JavaScript

Abstract

Due to the high rates of crimes such as theft and robbery, new technologies tend to be developed. With the popularity of the smartphone and the importance of its applications, in recent years its use has become almost indispensable on people's lives. Personal vehicle monitoring can become one of the main applications' functions, being used to help avoid reactions by vehicle owners in any robbery, informing the location of the internally installed security device. Report events such as a potential invasion and vehicle access, and to facilitate access by using the owner's smartphone for permission, are also features that can keep the user aware of the vehicle's situation at any time. In order to develop a security system with mobile application interaction with a device that can be easily installed, this project has been developed. This work, besides the physical device implementation, it includes the development of an application on the mobile iOS platform and a cloud service for being consumed by the application and the installed device inside the vehicle. With these parts integration, it was possible to build a system that informs the user about physical events and location. Instabilities were detected on many tests and were resolved using capacitors and appropriate power supply with enough current to all the modules. These instabilities were noted mainly in the WiFi module operations, which proved to be a very sensitive device. The use of MBaaS service allowed the implementation of many features to be done with less effort, summarizing the system needs in simple interactions with the sevice's application, and implementation of all the system's logic using just a JavaScript file to make the whole interaction with the iOS application and the physical device. The quality and the type of the PIR sensor is an important system factor, requiring most studied techniques to remove false alarms of potential intrusion.

Palavras Chave: Smartphone, iOS, GSM, WiFi, MBaaS, Cloud, Arduino, PIR Sensor, APNS, Objective-C, JavaScript

1 Introdução

1.1 Apresentação do Problema

Levantamentos estatísticos do 9ž Anuário Brasileiro de Segurança Pública, tabulado pelo Fórum Brasileiro de Segurança Pública (FBSP), mostra que Porto Alegre, Salvador e São Paulo são as capitais que apresentam os maiores índices de roubos de carros para cada 100 mil veículos (HISAYASU, brasil.estadao.com.br, 2015).

Segundo dados provenientes de empresas seguradoras, consolidados pela (SUSEP, www.susep.gov.br, 2016), Superintendência de Seguros Privados, o aumento do número de veículos segurados no país aumentou 7,1% no primeiro semestre do ano de 2014, em comparação com o mesmo período do ano anterior, sendo o Distrito Federal o estado de maior aumento no índice, com 44,4%, tendo em Brasília um total de 42.788 veículos roubados de uma frota segurada de 2.790.634 em 2014. Dados mais recentes divulgados pela (SSP-RS, www.ssp.rs.gov.br, 2016), Secretaria de Segurança Pública do Rio Grande do Sul, informam que roubos de veículos tem alto aumento de 31,85% no ano de 2015.

Novas tecnologias e sistemas antifurto tendem a serem criadas, envolvendo rastreamento por GPS, comunicação automática com a polícia e bloqueadores antifurto. O uso de smartphones tem aumentado a cada dia, são usados como ferramentas de trabalho lazer e comunicação. O smartphone é um tele móvel com funcionalidades avançadas que podem ser estendidas através de programas e aplicações executadas no seu sistema operacional. (BRAZ, www.mediassociais.com, 2012) Com o crescimento da tendência da utilização dos smartphones, dispositivos eletrônicos comuns a grande parte da população podem tornar a monitoração de um carro pessoal mais interessante no ponto de vista de comodidade e confiança. Como aproveitar as funções de um objeto de comunicação com tantas utilidades, e tão conhecido e necessário no dia a dia das pessoas, para a segurança de seu veículo, um bem com custo relativamente alto e também bastante necessário na vida de muitas pessoas?

1.2 Objetivos do Trabalho

1.2.1 Objetivos Gerais

O objetivo principal deste trabalho é implementar um sistema monitorador e facilitador de acesso ao veículo. A monitoração tem conexão com a Internet, com o objetivo de manter o dono do veículo ciente de movimentos suspeitos no interior do veículo e a sua localização a qualquer momento.

1.2.2 Objetivos Específicos

- Conhecer, projetar e implementar a programação dos módulos que atuam no sistema como controle de automação.
- Implementar a aplicação para smartphone para controle remoto do dispositivo.
- Projetar e montar conexões físicas entre os módulos, comunicação entre módulos e smartphone e interação com o serviço acessado pela Internet.

1.3 Justificativa e Importância do Trabalho

Em um artigo sobre a importância dos aplicativos móveis, o autor enfatiza a crescente informatização no cotidiano das pessoas, pelo fato de utilizarem frequentemente aplicativos das mais diversas categorias com os mais variados recursos para facilitar o dia a dia. Os smartphones ganharam espaço juntamente com os aplicativos tratados como seus "acessórios"a partir de 2007, ganhando espaço na vida das pessoas, assumindo tarefas simples do dia a dia, segundo o autor (MALUF, noticias.r7.com, 2015).

Como uma das funções de importância de aplicativos, não só podem ser utilizados para fins de assegurar a integridade de um bem como o automóvel utilizando o recurso de localização, mas podem auxiliar também para evitar possíveis reações por parte dos donos dos automóveis no momento de eventuais roubos por criminosos, ocorridas pela falta de certeza se o veículo perdido será recuperado, e por não saber onde se encontra instantes após o ocorrido.

1.4 Escopo do Trabalho

O projeto abrange implementação de hardware e software dos módulos WiFi e GSM, que efetuam conexões para comunicação com fio e sem fio. Inclui também uma integração física entre os módulos citados e um controle de trava do automóvel que é necessária para seu acionamento pelo dispositivo do sistema de segurança.

Utilizando a conectividade automática com o smartphone do usuário, o sistema autoriza o acesso ao veículo de modo que controle as travas automáticas. O sistema, com funcionalidade configurável, autoriza mais de um dispositivo do usuário, para ter acesso ao veículo, ou desativar a permissão automática por aproximação, caso ocorra a perda de um de seus dispositivos autorizados, como seu smartphone pessoal por exemplo, tendo como alternativa utilizar outro dispositivo cadastrado, se houver.

O sistema utiliza o módulo GSM/GPRS conectado à Internet para buscar a lista de dispositivos autorizados do sistema, bem como enviar eventos detectados por sensor PIR e controle de acesso ao veículo.

O desenvolvimento da arquitetura e do código do aplicativo na plataforma móvel iOS, que provê configurações do sistema e o serviço web ao qual o aplicativo e o dispositivo de segurança se conectam, serão objetos de estudo deste trabalho.

1.5 Resultados Esperados

- Trava automática: espera-se que quando o usuário se aproxime do veículo, as travas elétricas se destranquem. O alcance do módulo WiFi deve ser modificado para alcançar esta distância, necessitando de testes com técnicas de limitação de alcance do sinal WiFi.

- Detecção de presença: um sensor de presença PIR invocará o módulo GSM para enviar comandos para o servidor web sobre alguma movimentação suspeita no banco do motorista, gerando o envio de notificações, caso for necessário.

- Registros de localização: registros de localização estipulada por triangulação por função nativa do módulo GSM, deverão ser salvos no servidor web. É interesse deste trabalho o registro de localização a qualquer momento que seja requisitado pelo usuário.

- Conexão à Internet: a principal função do módulo GSM é a de conexão com a Internet, por prover localizações e receber comandos importantes.

- Notificações: notificações para informar ao usuário sobre algum evento suspeito são enviadas para o seu celular.

- Aplicativo: o aplicativo é responsável por receber informações do servidor web, que coleta localizações do módulo GSM/GPRS. As seguintes ações podem ser acionadas utilizando-se do aplicativo quando o smartphone estiver conectado ao módulo WiFi do dispositivo de segurança: acionar trancar ou destrancar o veículo, solicitar sincronização da lista de dispositivos autorizados do sistema e solicitar envio da localização do dispositivo no momento.

  São configurações do sistema controladas pelo aplicativo: ativar ou desativar o envio de notificações sobre eventos referentes ao sistema; Ativar ou desativar comandos de acionamento da trava elétrica do automóvel e; Ativar ou desativar funcionamento do sensor de presença PIR.

1.6 Estrutura do Trabalho

Este trabalho é dividido em seis capítulos. Os seguintes assuntos são apresentados nos próximos capítulos:

- Capítulo 1: Como aqui apresentado, com a introdução, apresentação do problema, objetivos, justificativa e importância do trabalho, escopo do trabalho e resultados esperados, assim como a estrutura do trabalho.

- Capítulo 2: Apresenta os problemas que motivam a pesquisa e desenvolvimento do projeto, através de estatísticas publicadas, estudos e levantamentos obtidos em pesquisas com os mesmos objetos de estudo e assuntos relacionados. São apresentados benefícios provenientes da implantação do projeto e consequências e riscos da não utilização.

- Capítulo 3: Técnicas, tecnologias, ferramentas e métodos utilizados no desenvolvimento do projeto, com detalhes que mostram os motivos pelos quais foram adotados são descritos neste capítulo, visando resolver os problemas propostos no primeiro capítulo.

- Capítulo 4: Este capítulo descreve de maneira geral o modelo proposto, detalhando as etapas, ferramentas, métodos e técnicas para a realização da implementação.
- Capítulo 5: Mostra-se os resultados e detalhes sobre dificuldades e etapas com certo nível especial de complexidade ou de esforço necessário.
- Capítulo 6: Conclui-se os resultados gerais sobre o projeto e sugestões para trabalhos futuros.

2 Apresentação do Problema

Estatísticas criminais apresentadas no sítio do Sistema Nacional de Informações de Segurança Pública, Prisionais e sobre Drogas (SINESP, www.ssp.rs.gov.br, 2016) retratam a situação de segurança pública contendo dados provenientes dos órgãos de Segurança Pública nas Unidades de Federação desde o ano de 2004. Dentre os relatórios de tipos de crimes específicos, estão os relativos à ocorrências que envolvem furtos e roubos de veículos. Em relação aos furtos de veículos automotores terrestres sem carga transportada, ou seja, automóveis de passeio, táxis, caminhonetes e motocicletas, por exemplo, foram obtidos dados nos anos de 2011, 2012, 2013 e 2014, onde demonstra a evolução da taxa de furtos de veículos por 100 mil veículos neste período. A figura 2.1 mostra um gráfico de número de ocorrências de furtos de veículos registrados entre 2011 e 2014. Nos anos de 2013, foram apresentados aproximadamente 307 veículos por 100 mil veículos furtados e no ano de 2014, apesar de haver uma queda na taxa para aproximadamente 284 veículos, nota-se que em relação aos anos de 2012 e 2014, ainda há um aumento significativo, tornando 2014 um ano de recuperação da alta variação ocorrida no ano de 2013. No gráfico de roubos, mostrado na figura 2.2, têm-se também um semelhante aumento na taxa do ano 2012 ao ano de 2014, mesmo com queda alta, entre 2013 e 2014.

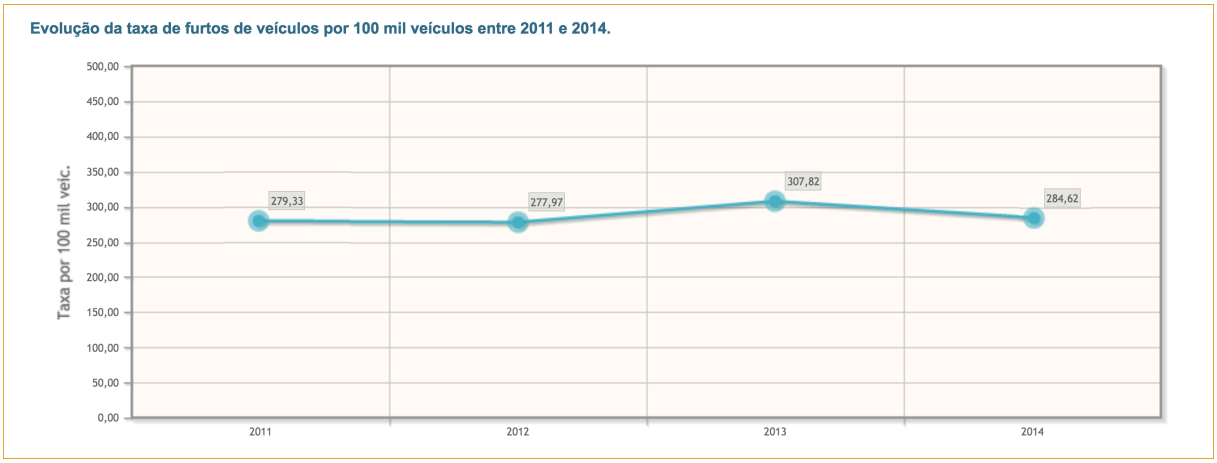


Figura 2.1 – Relatório Consolidado de Ocorrências de furtos de veículos registrados pela Polícia Civil. - SINESP - Disponível em `https://www.sinesp.gov.br/estatisticas-publicas` Acesso em 31/03/2016

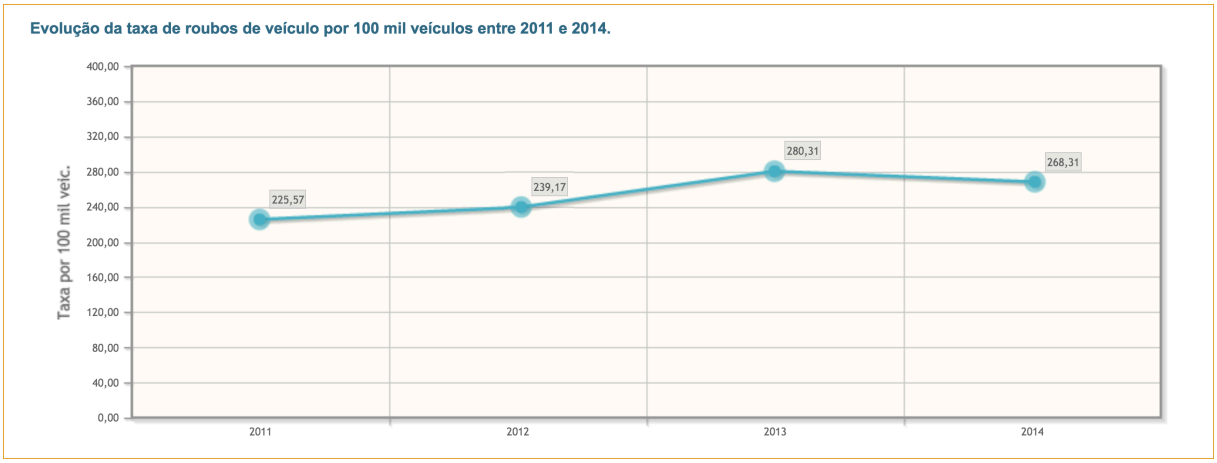


Figura 2.2 – Evolução da taxa de roubos de veículo por 100 mil veículos entre 2011 e 2014. - SINESP - Disponível em `https://www.sinesp.gov.br/estatisticas-publicas` Acesso em 31/03/2016

A diminuição da quantidade de crimes desta natureza estão nas mãos das tecnologias que vem evoluindo desde a invenção do veículo terrestre. Técnicas e componentes feitos para se instalar, como alarmes, travas de volante e pedais, por exemplo, foram utilizados ao passar do tempo, tentando sempre inibir os criminosos, que almejam estes bens que em sua maioria nos dias de hoje, bens de alto custo, sendo esta valorização o principal motivo pelo qual tantos crimes do tipo acontecem.

As chaves de carros sempre foram itens importantes para o acesso e o funcionamento do veículo. Antes elas não eram capazes nem de acionar o motor, porém nos últimos 100 anos teve grande evolução, podendo atualmente até estacionar um carro à distância. A primeira marca a utilizar a tecnologia de acionamento de motor utilizando uma chave, foi a Mercedes-Benz, que na década de 90 tornou-se possível dirigir um modelo de um veículo da marca com a chave no bolso o tempo todo (CHIPTRONIC, chiptronic.com.br, 2016).

Algumas tecnologias que foram criadas ao longo do tempo, foram adotadas pelos donos de carro como indispensáveis. Uma das mais difundidas na época em que foram trazidas como novidades pelas suas técnicas anti-furto, é a trava de câmbio de uma empresa fundada por dois israelenses em 1973, chamada MUL-T-LOCK (MUL-T-LOCK, www.multlock.com.br, 2016). O objetivo desta técnica, utilizada até os dias de hoje, seria bloquear movimentos na parte do câmbio do carro, evitando assim que criminosos conseguissem dirigir o veículo. Atualmente podem ser destrancadas so-

mente com uma chave codificada por computador. Na época em que ficou conhecida, por volta das décadas de 80 e 90, supria a necessidade em um mercado onde alarmes eram caros e raros.

Os alarmes automotivos, bastante usados atualmente, tem como objetivo principal impedir que o veículo ou os pertences em seu interior sejam furtados (GIOVANNETTI, maua.br, 2011). Para chamar atenção próximo ao veículo como um alerta, evita-se que o criminoso consiga agir, emitindo um alto som e piscando as luzes dos faróis do veículo. Os primeiros alarmes criados restringiam-se apenas a soar o alarme do veículo quando uma porta fosse aberta.

Atualmente os alarmes mais comuns são compostos de vários sensores, como os de ultrassom que identificam agitações internamente, medindo a intensidade dos movimentos detectados, emitindo o alarme caso os níveis ultrapassem certa faixa. Os sistemas de alarme são também compostos por (GIOVANNETTI, maua.br, 2011):

- Uma sirene capaz de criar vários sons, para que se possa escolher, com o intuito de identificar seu veículo pessoal;
- Um receptor de rádio que permite controle sem fio a partir de um chaveiro;
- Uma bateria auxiliar para que o alarme funcione mesmo quando a bateria principal estiver desconectada;
- Uma unidade de controle que recebe as informações dos sensores, faz controle geral e soa o alarme.

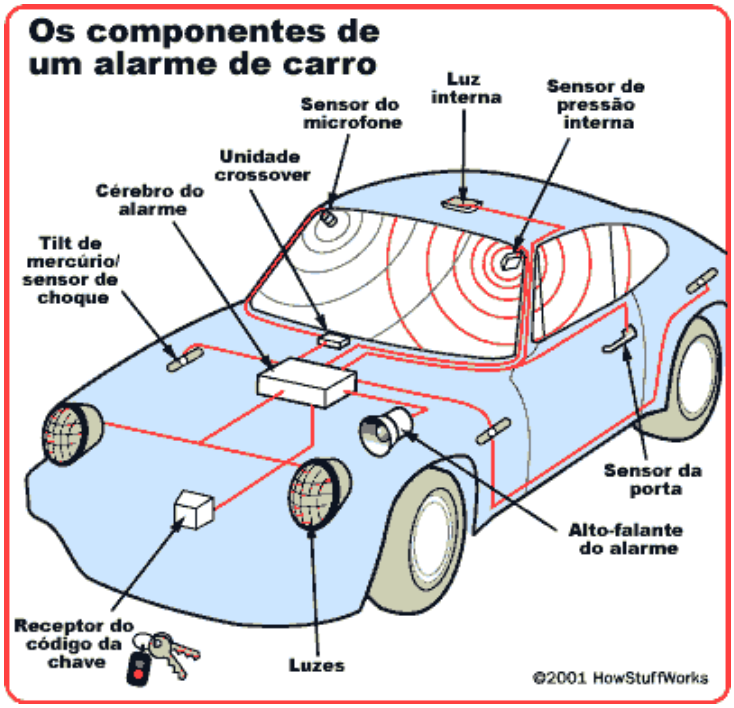


Figura 2.3 – Como funcionam os alarmes de carros - HSW - Como tudo funciona - Disponível em `http://carros.hsw.uol.com.br/alarmes-dos-carros.htm` - Acesso em 09/04/2016

Sistemas mais modernos combinam técnicas como detecção de movimentos, sensores de pressão e detecção de luz e presença, para que juntas tornem a proteção mais inteligente. A figura 2.3 mostra vários componentes de um sistema de alarme, integrados de maneira que juntos obtenham uma detecção de intrusão ou tentativa de violação eficiente.

Os rastreadores para carros são equipamentos de geolocalização que indicam o histórico e a localização do veículo quando estão circulando ou estacionados. Rastreadores funcionam utilizando o sistema de radiofrequência ou o GPS. O funcionamento via radiofrequência é baseado na triangulação de sinais captados por redes de antenas receptoras. O GPS funciona através de uma rede de satélites que circulam em volta da Terra. (BIDU, 2016).

Atualmente os equipamentos rastreadores são o principal meio para se monitorar a situação geográfica de um veículo furtado ou roubado, podendo assim tomar providências de busca, gerando uma grande chance de se recuperar o veículo. Os localizadores informam onde o carro furtado ou roubado está localizado no momento em que o motorista aciona a central de monitoramento, não possuindo históricor e acompanhamento oferecido pelo rastreador.

A combinação de várias destas soluções contra furto e roubo de objetos no interior e do próprio veículo aumentam a segurança, de forma que evitam estes tipos de violação simplesmente por serem identificadas por um criminoso, e por conhecerem as consequências de certas soluções como as de rastreamento, que contatam a polícia e de desligar o veículo ao detectar alguma irregularidade.

O sistema desenvolvido neste projeto visa utilizar algumas técnicas para especificamente evitar perigo quando o dono do veículo se aproxima para entrar no carro, de modo que utilize seu smartphone como ferramenta para autorizar sua entrada, e que tranque o veículo automaticamente, evitando eventual esquecimento, o que favorece ações criminosas. O sistema tem o intuito também de monitorar e informar ao usuário sobre movimento suspeito no interior do carro na parte do banco do motorista, enviando imediatamente um aviso ao seu smartphone, informando sua localização e o motivo do aviso.

O desafio em questão, questiona como tornar possível desenvolver um sistema que permita acionar as travas e o alarme do veículo automaticamente e ao mesmo tempo enviar avisos ao usuário, mantendo-o ciente de qualquer evento suspeito.

3 Referencial Teórico e Bases Metodológicas

Este capítulo apresenta as tecnologias, técnicas e conceitos utilizados para a implementação de cada componente atuante do funcionamento do sistema. Os módulos, cujas funções são de maior importância, terão suas implementações e características detalhadas inicialmente, bem como os meios em que se comunicam com a unidade de controle, que seria a placa Arduino, detalhada também com o mesmo nível dos módulos. O sensor de movimento e componentes físicos que integram o circuito montado juntamente com os módulos conectados, têm suas características, especificações e utilizações descritas, com o intuito de destacar detalhes mínimos que foram necessários para o funcionamento correto destas partes neste caso específico em que o sistema se encontra e foi planejado. A aplicação tem sua implementação detalhada no capítulo 4, mostrando as ferramentas utilizadas, a estrutura do projeto, compatibilidade em dispositivos, e requisitos mínimos para a construção e funcionamento do aplicativo móvel.

3.1 Comunicação Serial

As comunicações serais são sequências digitais de uns e zeros enviadas por um fio simples. Os protocolos para os dados enviados pelas linhas podem variar, mas os tipos de comunicação serial mais comuns são RS232, RS422 e RS485. O termo RS usado nessas denominações é um acrônimo para recommended standard (padrão recomendado). As comunicações RS232 utilizam linhas separadas para enviar e receber dados. Elas são conhecidas como TX para transmissão e RX para a recepção. As linhas TX e RX podem conectar-se nas extremidades de um mesmo pino, ou podem se conectar como RX-TX e TX-RX, o que é conhecido como configuração de modem nulo (null modem) (LAMB, 2015).

3.1.1 FT232R

O dispositivo FT232R é um circuito integrado de conversão entre USB e UART, Universal Asynchronous Receiver/Transmitter. Desta forma o microcontrolador envia as informações através de seu módulo UART e o FT232R envia ao computador pessoal

pela porta USB, tornando o protocolo USB praticamente transparente para o desenvolvimento do protótipo (SBORGI, D. & GIRIBONI, T., 2008).

O FT232R é fabricado pela empresa escocesa FTDI, Future Technology Devices International, criadora de dispositivos semicondutores, especializada em tecnologia USB.

3.2 Conjunto de Comandos Hayes (Comandos AT)

O conjunto de comandos Hayes (Hayes Standard AT command set), conhecido como comandos AT, é uma linguagem de comandos desenvolvida por Dennis Hayes e Dale Heatherington, para ser utilizada no modem Hayes Stack Smartmodem 300 da fabricante Hayes Microcomputer Products, em 1981. Este marcante dispositivo chamado de modem, (de ModuladorDemodulador) anunciou uma nova era das comunicações, e desempenhou um importante papel na explosão da Internet (DALAKOV, 2016).

O modem tornou-se viável para o usuário comum a partir do início da década de 1980. A partir de então, os modems passaram a evoluir juntamente com os computadores (PARANHOS, 2013).

Quase todos os comandos desde conjunto começam com as letras "AT", que seria um trecho inicial de um comando para primeiro chamar a atenção do modem. Por isso, são chamados de comandos AT, que até hoje ainda são usados em muitas das extensões criadas em conjuntos de comandos de fabricantes atuais. Estes comandos foram criados com o intuito de prover comunicação entre modems e dispositivos, tornando possível ações como definir a taxa de transmissão do modem, configurar alguns aspectos do dispositivo, enviar uma mensagem SMS e até iniciar ou atender uma ligação telefônica, por exemplo, através do sistema de telecomunicações móveis GSM. Alguns desses comandos são listados na tabela 3.1 (USRobotics, 2016).

Tabela 3.1 – Listagem de alguns comandos AT utilizado em modems

Comando	Sintaxe	Função
DT	ATDT	T Discagem por tom
DP	ATDP	P Discagem por pulso
DR	ATDR	R Chamar apenas um modem de origem
DL	ATDL	Rediscar o último número
L1	ATL1	Ligar o volume do alto-falante do modem
&K0	AT&K0	Desativar compactação de dados
&N6	AT&N6	Velocidade do Link - 9.600 bps
&Zn=s	AT&Zn=s	Armazenar número telefônico
&Zn?	AT&Zn?	Exibir número telefônico

3.3 Módulo WiFi ESP8266 ESP-01

O módulo ESP8266, que é originalmente fabricado pela Espressif, possui funções de comunicação por tecnologia Wi-Fi. Este módulo possui baixíssimo custo, e um tamanho bastante reduzido (comumente encontrado por US$ 5,00), podendo ser utilizado em diversas aplicações, oferecendo uma interface Wi-Fi que permite conexão com outros dispositivos. Possui uma grande facilidade para se integrar a outras soluções, podendo utilizar também comunicação UART. Foi criado em diversas variantes por sua fabricante, sendo 12 tipos diferentes (IMOBILIS, 2016). A figura 3.1 ilustra o modelo ESP-01. O núcleo de sua CPU é baseado em um processador IP Xtensa customizável, da empresa Cadence, que foi modificado a critérios da Espressif (CURVELLO, 2015).

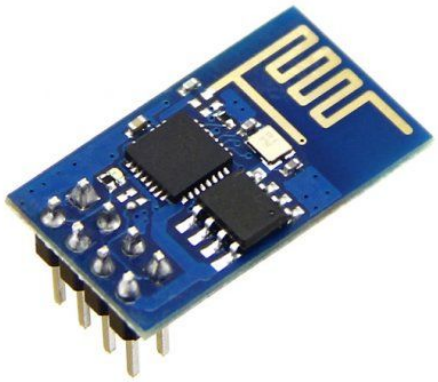


Figura 3.1 – ESP8266 Modelo ESP-01 fabricado pela Espressif. - iMobilis - Disponível em `http://www.decom.ufop.br/imobilis/esp8266-introducao-e-primeiros-passos/` Acesso em 31/03/2016

São listadas em seguida algumas das principais características deste módulo (IMOBILIS, 2016):

- Protocolo 802.11 b/g/n
- Wi-Fi Direct (P2P), soft-AP
- Pilha de protocolo TCP/IP integrada
- TR switch, balun, LNA e amplificador de potência integrados
- PLL, reguladores, e unidades de gerenciamento de energia integrados
- Potência de saída de +19.5dBm no modo 802.11b
- Sensor de temperatura integrado
- Suporte a diversas antenas
- CPU de 32-bit de baixo consumo integrada que pode ser usada como processador de aplicações
- SDIO 2.0, SPI, UART
- STBC, 1x1 MIMO, 2x1 MIMO
- Transmissão de pacotes em 2 ms
- Consumo em standby de 1.0mW (DTIM 3)

Alguns dos módulos vêm pré-carregados com um firmware que possibilita a comunicação por serial. Para realizar esta "Ponte Serial-WiFi", a interface serial do módulo obedece a uma lista ampla de comandos AT (CURVELLO, 2015).

3.3.1 Descrição da Pinagem

São mostrados na figura 3.2 os locais dos pinos do modelo ESP-01 do módulo, e sem seguida a tabela 3.2 lista os pinos da placa e sua respectiva função (CURVELLO, 2015).

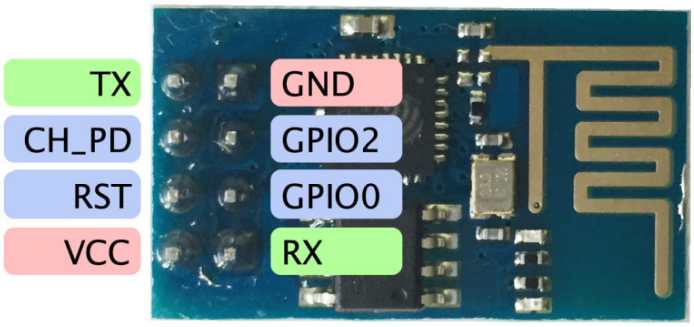


Figura 3.2 – Pinagem do ESP8266 modelo ESP-01. - Embarcados - Disponível em `http://www.embarcados.com.br/modulo-esp8266/` Acesso em 10/05/2016

Tabela 3.2 – Listagem de funções de cada pino do módulo ESP8266 modelo ESP-01.

Pino	Função
Vcc	Tensão de alimentação em 3,3V
GND	Sinal de terra
Tx	Sinal de saída de dados do módulo
Rx	Sinal de entrada de dados do módulo
RST	Sinal para reinicialização do módulo (Reset/Restart)
CH_PD	Sinal de habilitação do chip (chip enable), usado para gravação de firmware
GPIO0	Pino de comunicação entrada/saída que pode ser usado pelo firmware
GPIO2	Pino de comunicação entrada/saída que pode ser usado pelo firmware

3.4 Módulo GSM/GPRS SIM900a Mini

O módulo SIM900A Mini é um componente multifuncional baseado em um chip que funciona na rede móvel GSM/GPRS de celulares a dupla banda de 900/1800 MHz de performance para voz, SMS e dados, com um baixo consumo de energia (SIMCOM.eu, 2016). A figura 3.3 mostra o módulo SIM900A Mini e o chip SIM900A embutido na placa.

Figura 3.3 – Módulo GSM/GPRS SIM900A Mini com chip SIM900A embutido na placa (Autor)

Pelo fato de o modelo SIM900A (diferentemente do modelo SIM900), ser desenvolvido para ser utilizado em regiões da Ásia, é necessária a atualização para um firmware que a torna compatível com a rede brasileira e de outros países (PUJAR, www.raviyp.com, 2015).

3.4.1 GSM

O sistema GSM (atualmente, Global System for Mobile Communications) é um sistema celular digital de segunda geração, concebido com o propósito de resolver os problemas de fragmentação dos primeiros sistemas celulares da Europa. O GSM é o primeiro sistema celular do mundo a especificar modulação digital e arquiteturas de serviços de nível de rede (BRAGHETTO, L. F. B & SILVA, S. C.).

3.4.2 Cartão SIM

O cartão SIM é um circuito integrado do tipo Smart Card, que genericamente é um cartão com dimensões normalizadas pela norma ISO 7816. Os cartões SIM (Subscriber Identification Module) são utilizados pelo sistema de comunicações móveis GSM (Global System Mobile Communication). Alguns atores consideram estes cartões como um tipo de Smart Card (MAGALHÃES, 2003).

Um cartão SIM é internacionalmente identificado por seu circuito integrado de identificação de cartão, ICC-ID (Integrated Circuit Card Identifier), que é gravada no corpo

do cartão. É também identificado pela operadora com sua assinatura internacional de identidade móvel, IMSI (International Mobile Subscriber Identity). Essencialmente estes dois números são utilizados para a operadora identificar se o telefone celular é permitido para operar em sua rede e, quando conectado, deve ser cobrado por certas funcionalidades. (BADER, 2016).

3.4.3 Comunicação GPRS

O GPRS (General Packet Radio Service - Serviço de Rádio Geral por Pacotes) é um novo serviço *"novoice"* que permite que a informação em forma de dados seja emitida e recebida através de uma rede de telefonia móvel. Ele complementa os atuais serviços de comutação por circuitos GSM (Global System for Mobile) e os serviços de envio de mensagens via rede celular denominado de SMS (Short Message System) (BRAGHETTO, L. F. B & SILVA, S. C., 2003).

3.5 Arduino Pro Mini - Unidade de controle

Arduino é uma plataforma open-source de hardware usada para prototipação, baseada numa maneira fácil de se implementar hardware e software. As placas Arduino são capazes de ler entradas, e transformá-las em saídas. Essas entradas podem ser componentes como sensores, o apertar de um botão ou comandos recebidos de outro componente de hardware. As saídas podem atuar no controle de motores, diodos emissores de luz ou recebendo comandos de outros componentes de hardware. Intruções podem ser enviadas para o microcontrolador da placa utilizando a aplicação Arduino IDE e escrevendo código com a linguagem de programação Arduino. (ARDUINO, 2016)

O Arduino Pro Mini, mostrado na figura 3.4 é uma placa baseada no microcontrolador ATMega328. A placa contém 14 pinos digitais de entrada e saída (como mostrado na figura 3.5), sendo 6 possíveis de atuar como saídas PWM (modulação por largura de pulso): 1, 4, 5, 7, 12 e 11 mostradas na figura 3.5. Possui 6 entradas analógicas (5 a 8 e os dois furos próximos as entradas 5 e 6), um botão de reset, na parte esquerda da figura 3.4 e um ressonador (ARDUINO, 2016). Os pinos da esquerda são utilizados para comunicação serial com conexão por FTDI (placa para conversão de USB para TTL).

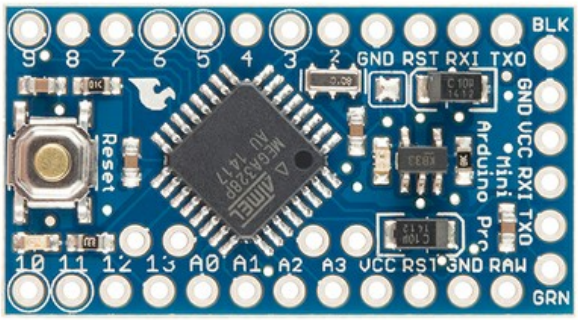


Figura 3.4 – Foto frontal do Arduino Pro Mini - Disponível em `https://www.arduino.cc/en/Main/arduinoBoardProMini` Acesso em 17/05/2016

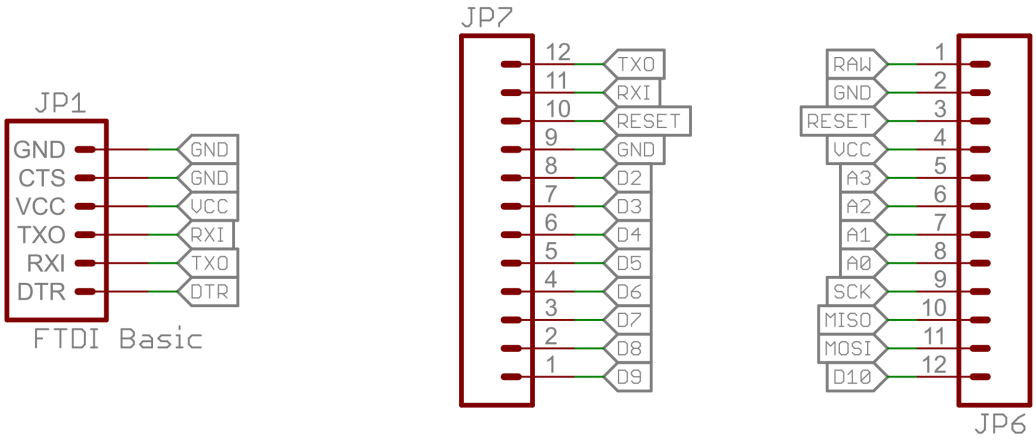


Figura 3.5 – Pinagem oficial do Arduino Pro Mini - Arduino Pro Mini Schematic - Disponível em `https://www.arduino.cc/en/uploads/Main/Arduino-Pro-Mini-schematic.pdf` Acesso em 17/05/2016

3.6 Sensor Infravermelho Passivo (PIR)

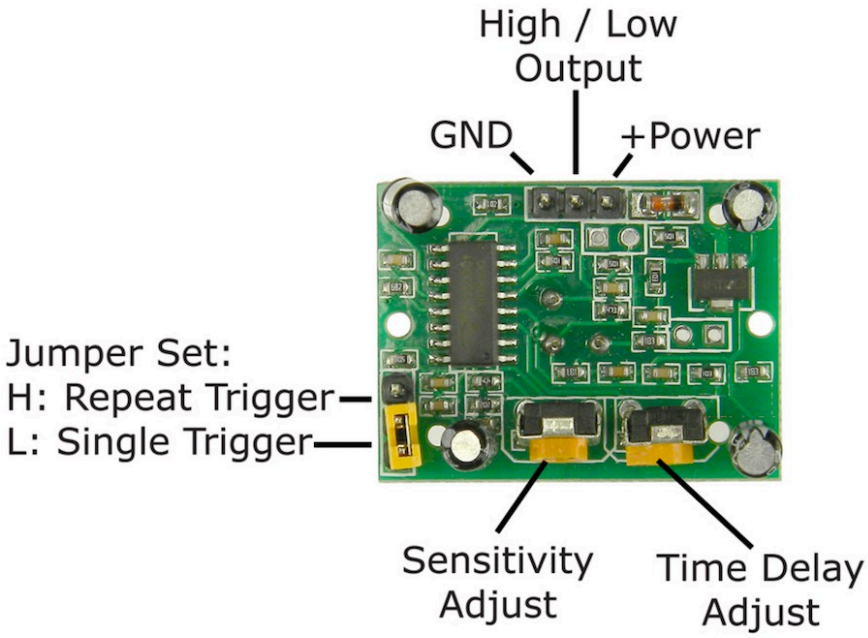


Figura 3.6 – Sensor de presença PIR HC-SR501 - Disponível em `https://www.mpja.com/download/31227sc.pdf` Acesso em 17/05/2016

Os sensores são dispositivos eletrônicos ou eletromecânicos responsáveis pela transformação de uma forma de energia em sinal elétrico. Os sensores infravermelho passivos são dispositivos eletrônicos que medem a luz infravermelha emitida dentro do seu campo de visão (LOUZANO, 2010). O termo passivo indica que o sensor PIR não emite nenhuma luz infravermelha, apenas recebe passivamente a radiação infravermelha do ambiente (TOREYIN, 2007). A figura 3.6 mostra o sensor PIR HC-SR501 e suas partes. O componente possui duas peças que rotacionam para ajustar a sensibilidade e o tempo de detecção.

3.7 Sistema operacional iOS

iOS, originalmente iPhone OS, é um sistema operacional móvel criado e desenvolvido pela empresa Apple, e é distribuído exclusivamente para dispositivos fabricados pela empresa, que são smartphones nomeados por iPhones, iPads, que são dispositivos com telas maiores e sem funcionalidades de telefonia, iPods, que possuem

características semelhantes aos iPhones porém também não possuem funcionalidades de telefonia. A figura 3.7 mostra os produtos iPhone e iPad, da empresa.

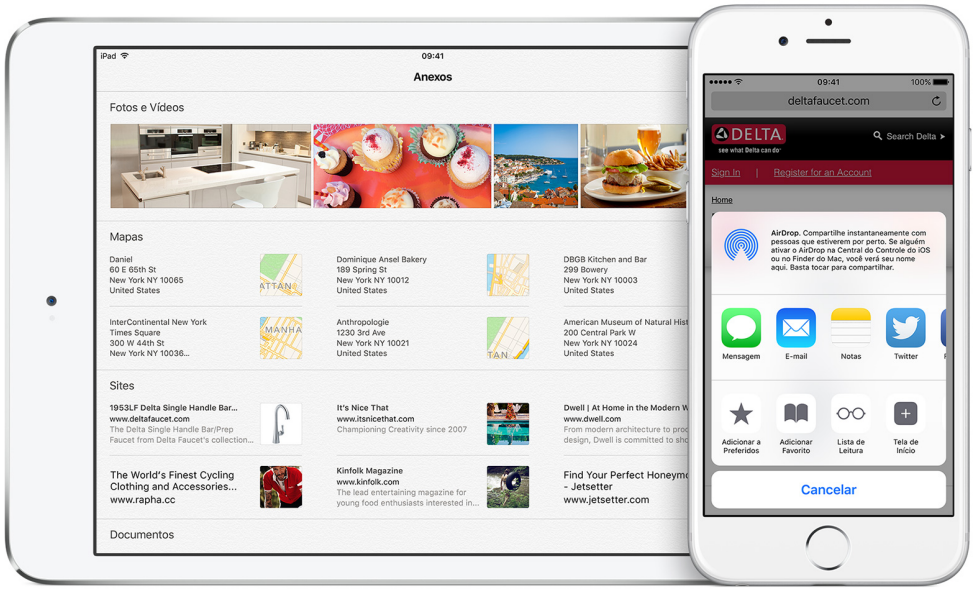


Figura 3.7 – O iPad (dispositivo maior) e o iPhone (à direita) são produtos que utilizam o sistema operacional iOS - Disponível em `http://www.apple.com/br/ios/whats-new/` Acesso em 11/05/2016

3.8 Apple Push Notification Service

Apple Push Notification Service (APNs) é um serviço de notificações remotas criado pela empresa Apple, para permitir que desenvolvedores de aplicações terceiras, enviem dados de notificação para aplicações instaladas em dispositivos Apple. O APNs é um sistema robusto e altamente eficiente de propagação de informação para dispositivos nas plataformas iOS (móvel, para celulares), watchOS (móvel, para relógios), tvOS (para televisores) e OS X (para computadores). Cada dispositivo estabelece uma conexão IP criptografada com APNs e recebe notificações por meio dessa conexão persistente (APPLE, developer.apple.com, 2016). Se uma notificação é recebida quando o aplicativo não está em funcionamento, o dispositivo alerta ao usuário que o aplicativo possui informações à sua espera. Um servidor é encarregado de gerar notificações remotas para seus usuários, conhecido como provedor, e reúne dados para seus usuários e decide quando uma notificação deve ser enviada. Para cada notificação, o provedor gera o corpo de dados e atribui este corpo a uma requisição HTTP/2, que envia para o APNs usando um canal persistente e seguro por protocolo múltiplo HTTP/2. Após a recepção do seu pedido, o APNs lida com a entrega do

corpo de dados ao aplicativo no dispositivo do usuário. A figura 3.8 mostra o caminho de uma notificação desde o provador até o aplicativo (APPLE, developer.apple.com, 2016).

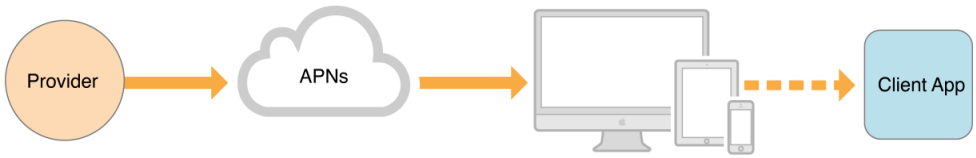


Figura 3.8 – Enviando uma notificação remota de um provedor para um aplicativo cliente - Disponível em `https://developer.apple.com/` Acesso em 03/05/2016

3.9 Solução MBaaS - Mobile Backend as a Service

Aplicações móveis podem necessitar de capacidades de armazenar dados em um servidor. Podem ser utilizadas por muitos usuários, que podem inserir informações colaborativamente, ou podem necessitar de habilidades para sincronizar dados para todos os dispositivos que um usuário utilizar para sua conta cadastrada no sistema.

A Mobile Backend as a Service (MBaaS), é uma tecnologia recente para facilitar o desenvolvimento de servidores voltados para o funcionamento juntamente com aplicações móveis. Um provedor MBaaS permite que usuários de seu serviço usem funcionalidades que necessitam de prévio desenvolvimento do chamado backend. As empresas que proveêm este tipo de serviço, entre outras estão: Microsoft Azure, Firebase, Parse e Kinvey (BENGTSSON, S. & ELIASSON, P., 2015).

3.9.1 Parse

O serviço MBaaS da empresa Parse, comprada pela empresa Facebook em 2013 (CUTLER, K. & CONSTINE J., 2013) possui as funcionalidades descritas anteriormente, e utiliza uma popular tecnologia de banco de dados tipo NoSQL chamada MongoDB para o armazenamento de dados. Provê também SDKs para multiplas plataformas, dentre elas Android, iOS e JavaScript (BENGTSSON, S. & ELIASSON, P., 2015).

O código Backend, chamado de Cloud Code no Parse, pode ser publicado em servidores hospedados pela empresa. Este código pode ser escrito para definir funções de API customizadas para lógica comum na aplicação ou perceber eventos e objetos

salvos ou excluídos, que podem ser utilizados para validar dados recebidos ou realizar ações quando dados são modificados (BENGTSSON, S. & ELIASSON, P., 2015).

3.10 Linguagem C/C++

A linguagem C foi desenvolvida para ser traduzida eficientemente para código de máquina rápido, com um mínimo de sobrecarga de manutenção. C++ foi construída sobre C, adicionando algumas características para "programação orientada a objetos", um estilo de programação que promete uma modelagem mais fácil de objetos do mundo real (HORSTMANN, Cay, 2008).

A versão inicial da linguagem C foi projetada por volta de 1972, mas novos recursos foram adicionados ao longo dos anos. O processo de projeto terminou em 1989 com a conclusão do padrão ANSI (American National Standards Institute). Neste meio tempo, Bjarne Stroustrup, da AT&T, adicionou a C características da linguagem Simula, resultando na linguagem denominada de C++. De 1985 até hoje, C++ tem crescido pela adição de diversos recursos, e um processo de padronização culminou com a publicação do padrão internacional de C++ em 1998 (HORSTMANN, Cay, 2008).

3.11 Linguagem Objective-C e IDE Xcode

A linguagem Objective-C foi criada por Brad Cox e Tom Love no início da década de 1980 e posteriormente, em 1988, foi adquirida e licenciada por Steve Jobs, que após um tempo foi utilizada pela Apple até os dias de hoje (LECHETA, 2016). Após começar a ser utilizada pela Apple, foram criados os frameworks AppKit, que dispõe de objetos necessários para implementação gráfica de interface de usuário (APPLE, 2014), e Foundation Framework, que define uma camada base de classes do Objective-C. (APPLE, 2013).

Objective-C é a linguagem primária utilizada para escrever softwares para os sistemas operacionais OS X, para computadores e iOS, para dispositivos móveis. É um superconjunto da linguagem de programação C e fornece capacidades orientadas a objeto e execução dinâmica. Objective-C herda a sintaxe, tipos primitivos, e declarações de controle de fluxo do C e adiciona sintaxe para definir classes e métodos. (APPLE, 2014). Pela influência da linguagem SmallTalk, o Objective-C foi criado com

o intuito de aprimorar a linguagem C, por apresentar o paradigma estruturado, sendo necessário utilizar o paradigma de orientação a objetos. (LECHETA, 2016).

3.11.1 Xcode

Xcode é a ferramenta de desenvolvimento IDE oficial da Apple, (LECHETA, 2106) e é utilizada para desenvolver projetos em OS X e iOS, para produtos Apple, incluindo iPad, iPhone, Apple Watch, e Mac. O Xcode, provê ferramentas para gerenciar inteiramente o processo de desenvolvimento, desde criar a aplicação, até testar, otimizar e enviar para a loja de aplicativos (APPLE, 2016). A figura 3.9 mostra a interface da ferramenta com suas principais divisórias:

- 'Utilities': possui opções para acessar ajuda rápida e escolher recursos prontos para se utilizar no projeto.
- 'Navigator': possui seções como visualização da raíz de diretórios, pesquisa de conteúdo nos projetos, avisos, erros e acompanhamento de execução dos aplicativos.
- 'Editor': área para codificação e construção de interface.
- 'Debug Area': área para visualizar logs, controla a execução e possui controle de navegação pelo código.

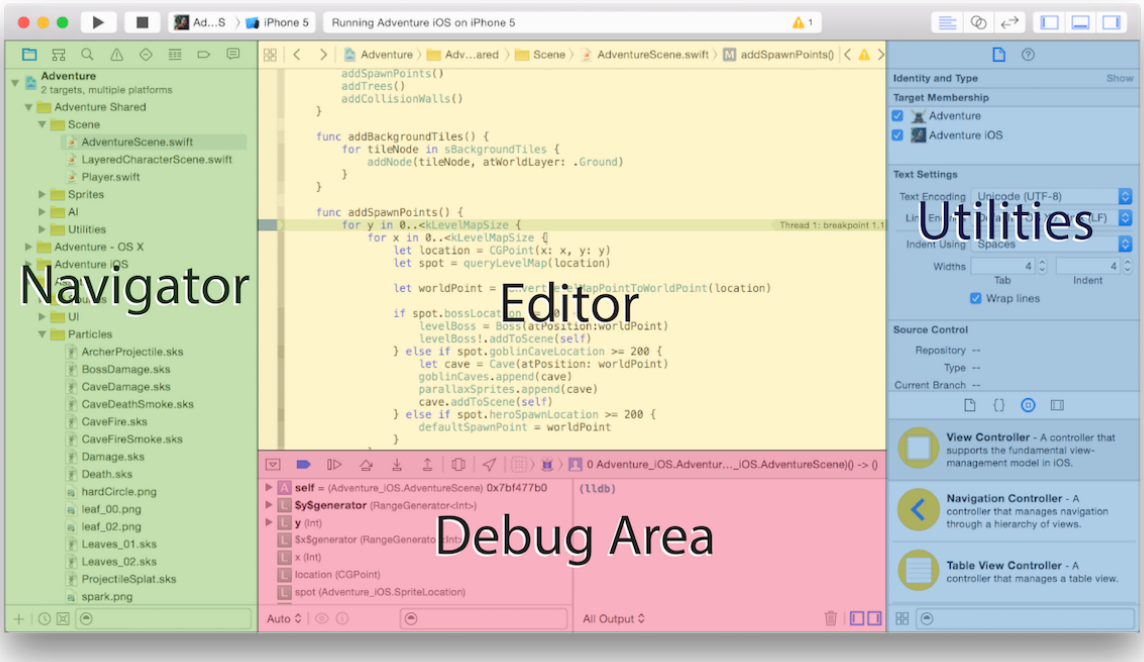


Figura 3.9 – Interface da IDE Xcode - Disponível em `https://developer.apple.com/library/` Acesso em 05/05/2016

O conteúdo técnico abordado neste capítulo são descritos para serem utilizados como base para o entendimento básico para possibilitar o desenvolvimento geral do projeto, que será detalhado no próximo capítulo.

4 Proposta de Solução para o Sistema de Segurança Veicular

4.1 Apresentação do Sistema

O sistema de segurança veicular, é composto de três tecnologias principais: armazenamento de informações em nuvem, instalação física no veículo e interação do usuário com o sistema utilizando aplicação em smartphone conectado à Internet, como mostra o esquemático da figura 4.1 (Autor).

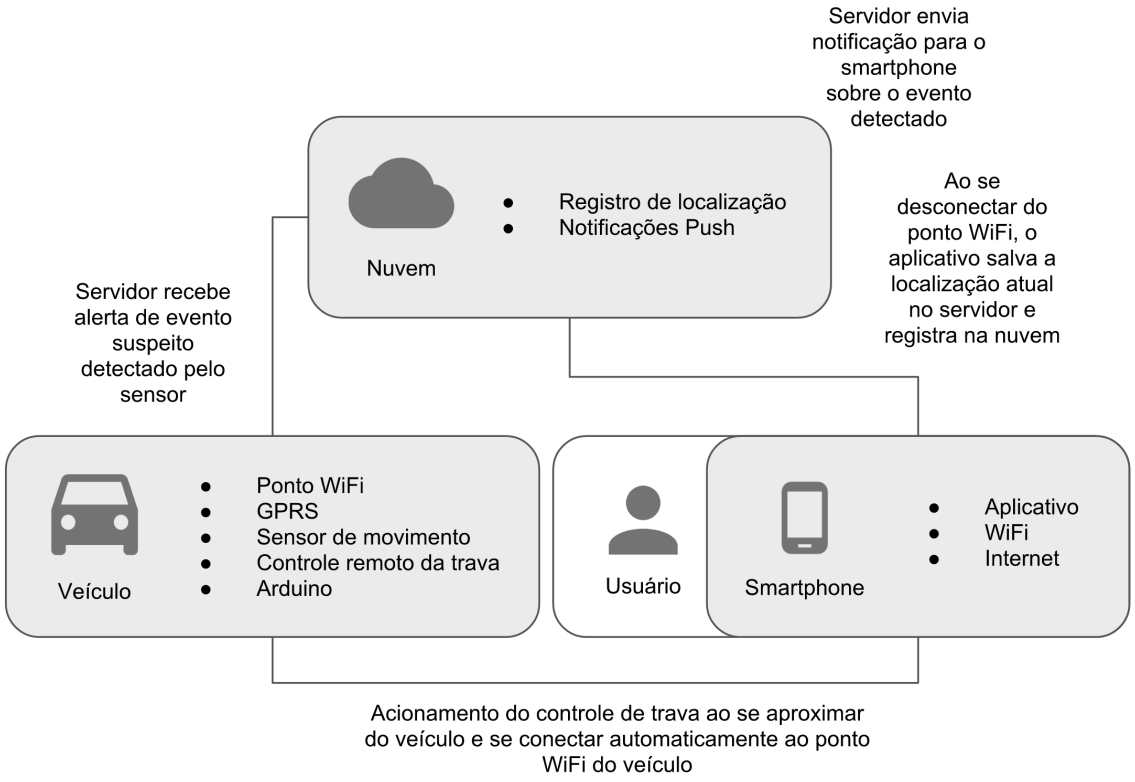


Figura 4.1 – Esquemático do sistema de segurança veicular

As etapas do projeto serão explicadas em detalhes. Para tanto, foram separadas várias partes que, ao final da implementação de cada uma delas, apresentam o sistema de forma integrada e em funcionamento.

4.1.1 Primeira etapa: construção do equipamento e implementação de hardware

A integração das três tecnologias principais do sistema é feita de modo que o funcionamento de cada parte seja fundamental. Porém, a parte primordial é a montagem

física do equipamento no veículo. Construindo e instalando o equipamento a partir da utilização e combinação de componentes, e com a implementação de hardware como primeira etapa, é possível realizar testes para obter bom feedback antes de avançar para o desenvolvimento das partes de interação por aplicativo e manipulação e armazenamento de informações. O equipamento do sistema é constituído pelos seguintes componentes: módulo WiFi, módulo GPRS/GSM, sensor de presença PIR, controle remoto de trava do veículo, placa Arduino Pro Mini, reguladores de tensão e componentes de controle. A implementação destes componentes para que trabalhem em conjunto, se dá por meio das seguintes ações:

1. Preparação dos módulos WiFi (ESP8266 ESP-01) e GPRS/GSM.
2. Preparação do Arduino Pro Mini.
3. Definição da comunicação entre a placa Arduino e os módulos WiFi e GPRS/GSM.
4. Implementação de detecção de dispositivos autorizados conectados ao módulo WiFi.
5. Preparação e integração do controle de travas automáticas.
6. Implementação de recebimento de comandos enviados pelo aplicativo.
7. Preparação e integração do sensor de presença PIR.
8. Implantação do dispositivo no interior de um veículo

4.1.2 Segunda etapa: implementação do serviço de comunicação e armazenamento de informações em nuvem

A segunda etapa do trabalho de construção do sistema propõe utilizar do funcionamento responsivo de parte da implementação física já realizada, para armazenar informações obtidas, como localização, eventos de trancamento e destrancamento do sistema de travas do veículo e detecção de presença pelo sensor PIR. A comunicação entre o sistema físico e o serviço de armazenamento e gerenciamento de dados, é feita a partir da conexão do módulo GPRS/GSM com a Internet, por meio de um canal de acesso ao serviço por conexão HTTP. O serviço disponibiliza um endereço (`pf-20973560.parseapp.com`) para prover informações para dispositivos externos. A

integração é realizada por meio de implementação do serviço MBaaS para armazenamento e manipulação de dados, criação das rotas de comunicação (situação do veículo (com localização), sincronização de endereços de dispositivos autorizados e eventos detectados pelo dispositivo).

4.1.3 Terceira etapa: desenvolvimento da aplicação para smartphone, interação com o sistema e recebimento de notificações

Por último, integra-se o canal com que o usuário interage com o sistema e recebe notificações dos eventos detectados fisicamente (e armazenados em nuvem). As funcionalidades da aplicação móvel em iOS, são constituídas em maior parte por chamadas ao serviço (MBaaS) em nuvem. Quando o serviço em nuvem recebe dados, deve-se tratar de modo que sejam salvos para serem utilizados pelo aplicativo, além de ser responsável pelo envio das notificações, gerando por isso, necessidade de implementações no ambiente do serviço em nuvem. O gerenciamento de dispositivos autorizados é feito por meio do aplicativo, adicionando, removendo e salvando a lista de dispositivos na nuvem para uso posterior do sistema físico, que também necessita de implementações adicionais para sincronizar esta lista. A interação com o sistema para acionamento de certas funções, se dá a partir de uma comunicação TCP/IP entre o smartphone e o módulo WiFi. A implementação torna finalizada quando se conclui a criação da aplicação, comunicação com o serviço na nuvem, envio de notificações para os dispositivos móveis e a interface do usuário.

4.2 Descrição das Etapas do Modelo

Nesta seção, são descritos os métodos usados no projeto desde o início até a conclusão do protótipo do sistema. Cada componente citado no capítulo anterior foi testado individualmente, considerando suas particularidades como base.

As etapas descritas a seguir seguem uma linha progressiva, sendo assim, ao concluir uma determinada etapa, torna-se possível o desenvolvimento da próxima, pois alguns testes são possíveis somente após a conclusão de atividades anteriores. Em seguida nas seções 4.2.1 a 4.2.9, são descritos os processos de impelementação de hardware e construção física do dispositivo de segurança.

4.2.1 Preparação do Módulo WiFi ESP8266 ESP-01

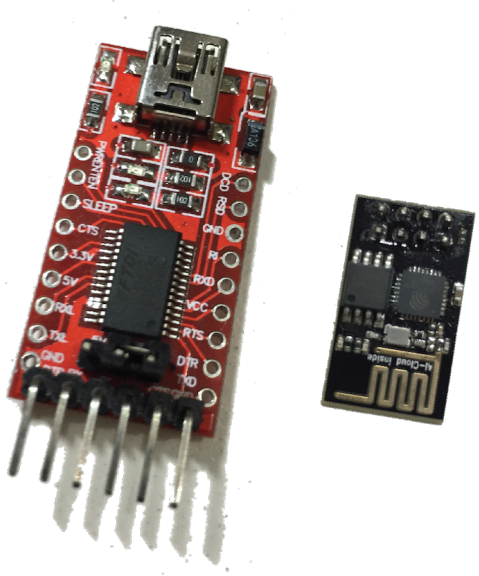


Figura 4.2 – Placa conversora USB para serial TTL à esquerda e módulo ESP8266 modelo ESP-01 à direita. (Autor)

A primeira etapa de iniciação da implementação física do projeto, foi a conexão e funcionamento da placa do módulo WiFi ESP8266 (mostrado ao lado direito da figura 4.2). Com relação às conexões da placa, deve-se utilizar a placa FTDI conversora de USB para serial TTL (mostrado ao lado esquerdo da figura 4.2), para permitir a comunicação do dispositivo com o computador utilizado como apoio para a implementação geral do projeto. As conexões mínimas para o funcionamento e teste básico da comunicação serial são as seguintes:

- Após certificar-se de que a placa FTDI possui um jumper para escolha da tensão ou pino para usar alimentação em 3,3V, conecta-se uma saída VCC da placa FTDI, com o pino VCC (em 3,3V) do ESP8266.
- Conecta-se o GND da placa FDTI na GND da placa ESP8266, para que ambos tenham a mesma referência de GND.
- Faz-se a ligação RX-TX entre as duas placas, ou seja, liga-se o pino RX de uma das placas no pino TX da outra, e vice-versa.
- O pino CH_PD da placa ESP8266, deve ser mantida em pull-up, ou seja, com um resistor de 10K Ohms ligando o pino ao VCC.

- Conecta-se a placa FTDI em uma entrada USB do computador.

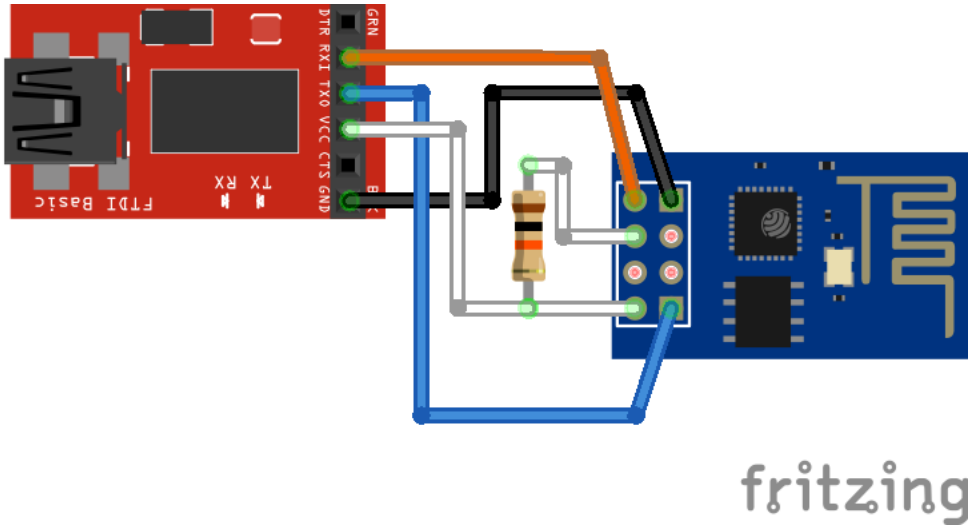


Figura 4.3 – Esquemático de conexões para funcionamento básico do ESP-01. (Figura gerada pela ferramenta Fritzing - `http://fritzing.org/`)

Pode-se observar na figura 4.3 o esquemático das conexões citadas nos itens acima. O pino CH_PD (chip power-down), tem como função ativar o funcionamento da placa. O pull-up utilizando resistor, mantém a entrada ajustada em nível lógico esperado, se o dispositivo for desconectado. A possibilidade de causar danos à placa por ligá-la em alimentação de tensão maior é considerável. Antes de realizar a ligação é recomendado medir a tensão do pino VCC para verificar seu valor.

Para realizar o teste de comunicação, pode-se utilizar qualquer versão da aplicação Arduino IDE, escolhendo a opção "Serial Monitor"no menu "Tools"ou "Ferramentas". Por padrão é provável que a placa tenha uma versão de firmware instalada e com taxa de transmissão de dados padrão em 9600 bps. Ao abrir a janela "Serial Monitor", escolhe-se a taxa de transmissão 9600 bps no canto direito inferior da janela, como mostra a figura 4.4.

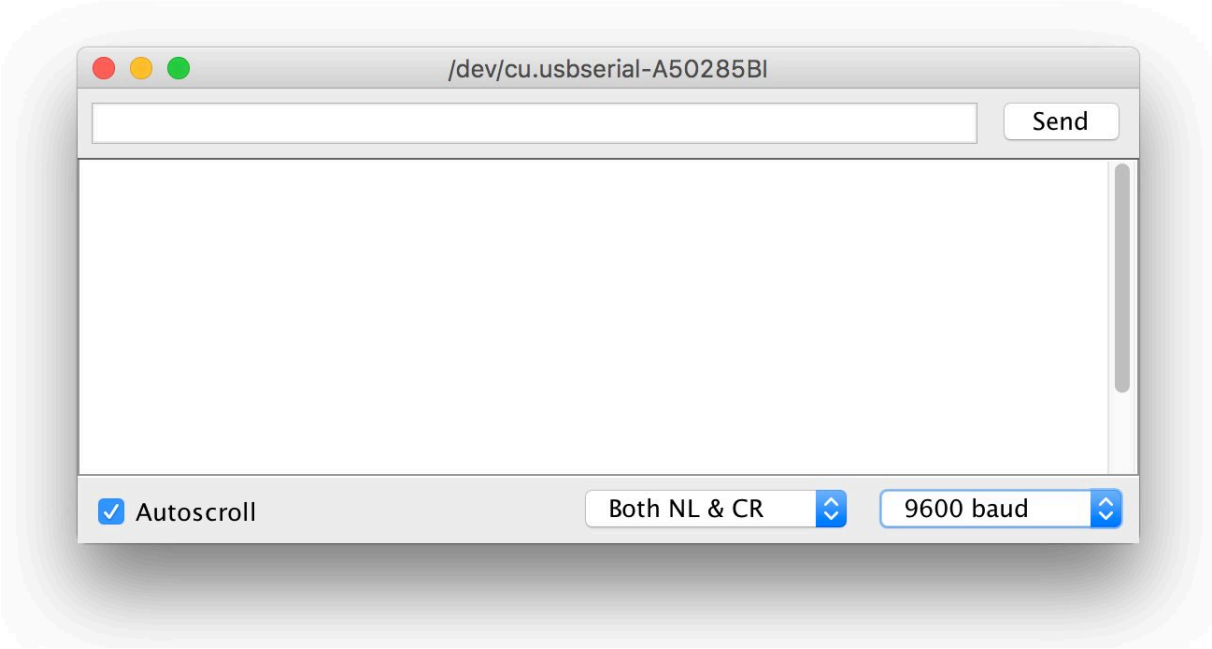


Figura 4.4 – Janela de monitoração serial da aplicação Arduino IDE. (Autor)

Após definir a taxa de transmissão, caso as conexões citadas acima estiverem corretas e o firmware já se encontrar instalado, as seguintes informações recebidas pela placa ESP-01 são mostradas na tela após o envio do comando 'AT+GMR':

```
AT+GMR
00200.9.5(b1)
compiled @ Dec 25 2014 21:40:28
AI-THINKER Dec 25 2014

OK
```

O comando 'AT+GMR' informa a versão do firmware instalado, sendo no caso mostrado, um exemplo de resposta que se pode obter com o firmware de versão 0.9.5(b1) (utilizado no projeto) instalado.

É possível verificar a versão atual do firmware instalado na placa utilizando o comando "AT+GMR". Espera-se que a versão 0.9.0.1 (é representada como "00160901"na resposta do comando citado) esteja instalada de fábrica (firmware). Existe a possibilidade de a placa necessitar de atualização de firmware, por não conter a versão de firmware 0.9.5(b1) utilizada no projeto. O firmware pode ser atualizado da seguinte forma:

Para realizar a atualização do firmware, é necessário que o módulo esteja conec-

tado da maneira descrita anteriormente, e que o pino GPIO0 (pode ser identificado na figura 3.2 do capítulo 3) do módulo esteja previamente aterrado, ou seja, antes de ligar a placa.

A ferramenta utilizada foi a chamada "esptool.py", um script na linguagem Python (criado por Fredrik Ahlberg e mantido atualmente por Fredrik and Angus Gratton) que comunica com o bootloader em memória ROM de módulos da série ESP8266. A ferramenta pode ser encontrada no seguinte endereço (Acesso em 18/05/2016):

`https://github.com/themadinventor/esptool/`

A ferramenta necessita da versão 2.7, ou mais recente instalada na máquina que se conecta o USB. Utiliza-se a opção "Download ZIP"na página do endereço citado da ferramenta, ou clone do repositório, utilizando o endereço do arquivo git informado na página (Acesso em 18/05/2016). O comando utilizado em terminal no sistema operacional OS X foi:

git clone `https://github.com/themadinventor/esptool.git`

Deve-se utilizar o seguinte comando para realizar a instalação da ferramenta no diretório em que seus arquivos foram descarregados:

python setup.py install

O firmware utilizado de versão 0.9.5(b1), pode ser encontrado no seguinte endereço: `http://abcdn1.qiniudn.com/ai-thinker-0.9.5.bin` (Acesso em 18/05/2016)

O arquivo foi colocado no mesmo diretório que a ferramenta "esptool.py", e o seguinte comando foi usado para executar o script para utilizar o arquivo do firmware:

```
python esptool.py -p /dev/tty.usbserial-A50285BI write_flash 0x000000 ai-thinker-0.9.5.bin
```

No comando acima, o trecho "python" executa o script "esptool.py" em seguida. O trecho "p", abreviação de "port" indica o caminho ("/dev/tty.usbserial-A50285BI") do dispositivo USB conversor FTDI utilizado para realizar comunicação serial. Os arquivos que representam os dispositivos USB podem ser identificados no diretório de dispositivos identificados pelo sistema: "/dev", tendo como início o trecho "tty.usb". O trecho "write_flash"indica o endereço de memória "0x000000"para o qual o firmware (de nome "ai-thinker-0.9.5.bin") será escrito.

A figura 4.5 mostra a execução do procedimento de atualização do firmware. Como informado pelo script, o processo levou pouco mais de 34 minutos (2070,1 segundos).

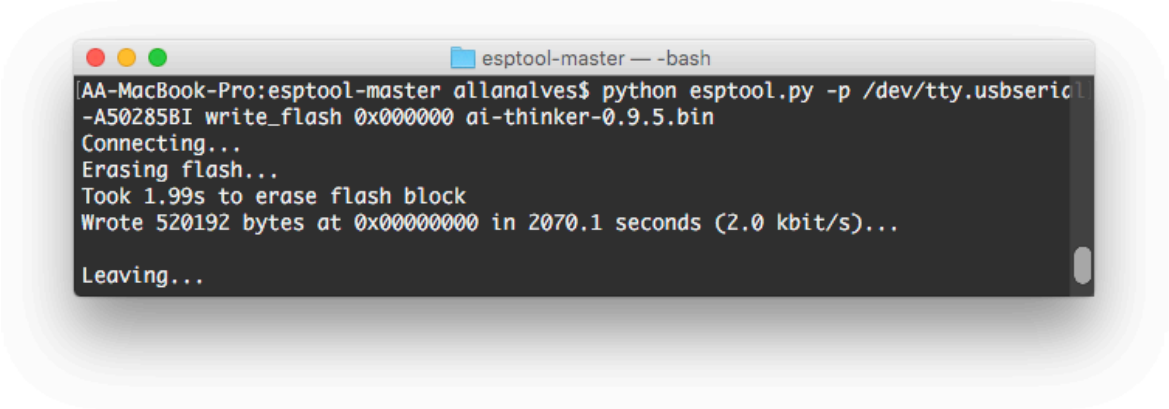


Figura 4.5 – Janela do terminal de comandos. Procedimento de escrita do firmware no módulo. (Autor)

A taxa de transferência de dados escolhida para se utilizar o módulo neste projeto é de 57600 bps. Para modificar a taxa atual, abre-se a janela de monitoração serial da aplicação Arduino IDE, seleciona-se a taxa de 9600 bps (taxa padrão após carregamento do firmware), e usa-se o seguinte comando:

```
"AT+CIOBAUD=57600"
```

4.2.2 Preparação da placa Arduino Pro Mini

São necessárias as seguintes conexões para preparar a placa e testar suas funcionalidades básicas:

- Com o conversor USB desconectado de qualquer outro componente, ertificar-se de que o jumper de seleção de tensão está posicionado em 5V, ou se há um pino para utilizar alimentação de 5V.
- Conectar as placas de maneira que o pino VCC do conversor ligue com o pino VCC da placa Arduino.
- Ligar o pino GND do conversor no pino BLK da placa Arduino.
- Ligar o pino CTS do conversor no pino GND da placa Arduino.
- Ligar pinos de transferência de dados RX e TX em seus respectivos pinos, como foi conectado o módulo WiFi anteriormente.

- Ligar o pino DTR do conversor no pino GRN da placa Arduino.

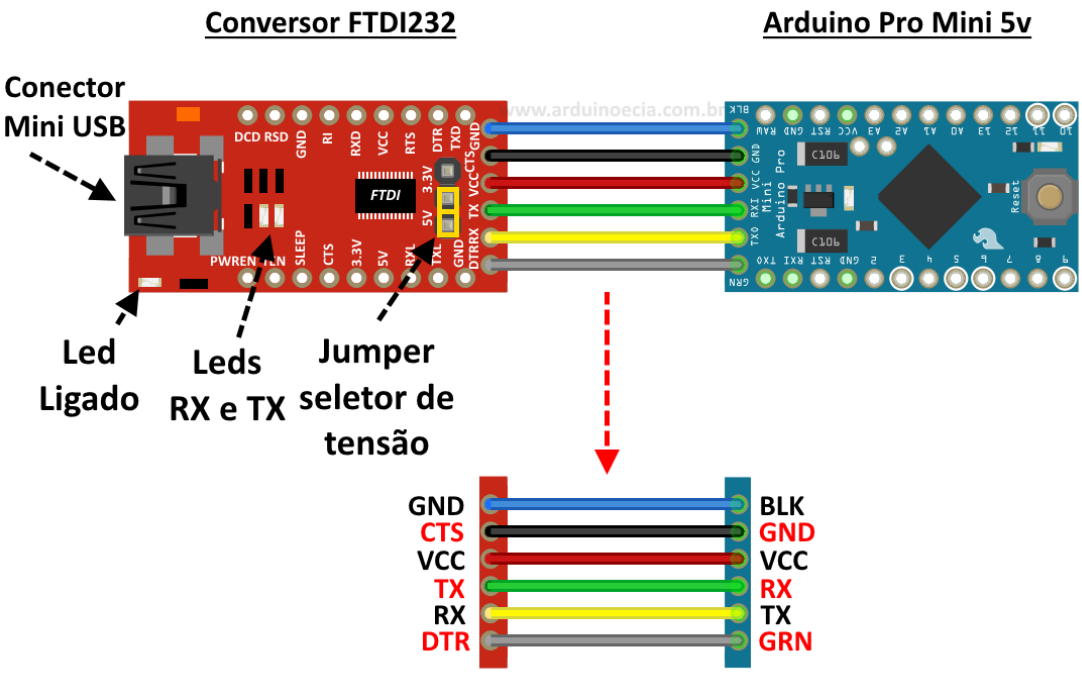


Figura 4.6 – Esquema de conexão da placa conversora USB com a placa Arduino Pro Mini. (www.arduinoecia.com.br - Acesso em 20/05/2016)

A representação dos procedimentos acima são mostrados na figura 4.6. Após as conexões, a placa conversora sinaliza seu funcionamento e a transferencia de dados ao acender dos LEDs indicados na figura 4.6.

Para testar o funcionamento da placa, pode-se utilizar a aplicação Arduino IDE. Ao iniciar a aplicação, seleciona-se a opção da placa "Arduino Pro or Pro Mini" no subitem "Board:" (placa) do menu "Tools"(ferramentas). No subitem "Processor:" seleciona-se o processador e suas características correspondentes ao modelo da placa utilizado no projeto: ATmega238 (5V, 16 MHz). A figura 4.7 mostra o menu "Tools" e seus subitens citados com as opções selecionadas descritas.

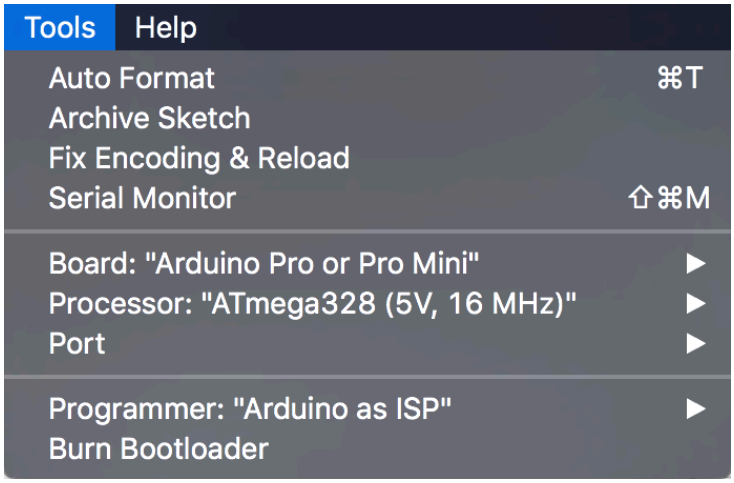


Figura 4.7 – Subitens do menu de ferramentas da aplicação Arduino IDE com suas opções necessárias selecionadas. (Autor)

Após conectar o conversor USB em uma porta do computador, verificar ainda no menu de ferramentas, se o dispositivo conversor é listado, selecionando-o para verificação do funcionamento da placa Arduino.

Quando prontas as etapas anteriores, abre-se um código básico de exemplo utilizando seguinte item de menu: File > Examples > 01.Basics > Blink (procedimento mostrado na figura 4.8). Ao selecionar, uma nova janela é aberta contendo o arquivo de exemplo, que pode ser compilado e carregado para a placa selecionando o item de menu: Sketch > Upload.

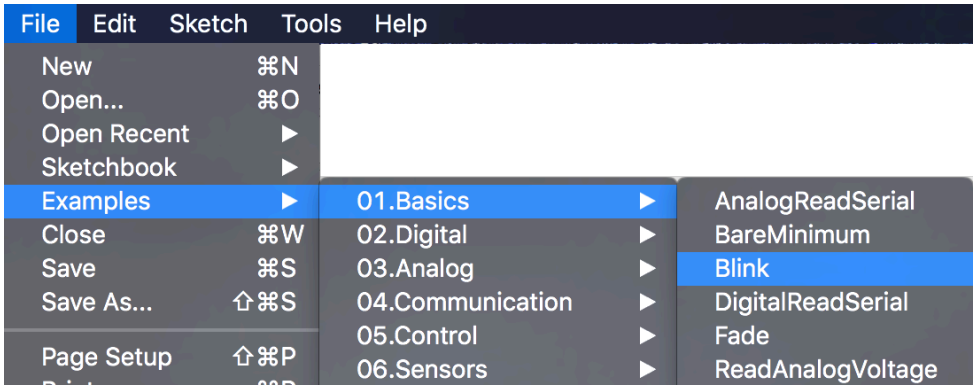


Figura 4.8 – Seleção de arquivo de código para teste da placa Arduino. (Autor)

Ao terminar de carregar o código para a placa, deve-se verificar se o LED embutido da placa pisca em um intervalo de 1 segundo. Caso ocorra, pode-se notar que a placa está em condição de funcionamento, concluindo assim, o procedimento de carregamento de código, que será repetido algumas vezes para implementações posteriores.

4.2.3 Definição da comunicação entre a placa ESP8266 e a placa Arduino Pro Mini

Assim que concluídos os testes utilizando o conversor e a funcionalidade de monitorar a comunicação serial entre os dispositivos, já torna possível conectar as placas WiFi e Arduino. Em seguida descreve-se os procedimentos de conexão:

- Do mesmo modo, conecta-se a placa Arduino com o conversor serial, com a tensão selecionada para 5V.
- Utilizando um pino de alimentação de 3,3V da placa do conversor serial, conecta-se diretamente ao VCC do módulo WiFi, fornecendo então a tensão adequada para a placa.
- Liga-se o GND do módulo ESP8266 na mesma referência de terra das outras placas, inclusive os pinos GND do conversor de nível lógico.
- Usa-se o pino de 3,3V para dar referência ao conversor de nível lógico, ligando em LV.
- Conecta-se o pino de saída de 5V no pino HV do conversor de nível lógico.
- Conecta-se os pinos TX e RX do módulo ESP8266 nas entradas RXO e TXI respectivamente.
- Liga-se os pinos 3 (RX) e 4 (TX) da placa Arduino nos pinos RXI e TXO que dá conexão aos pinos TX e RX da placa ESP8266.
- Como feito na etapa de preparação do módulo WiFi individualmente, conecta-se um resistor de 10K Ohms como pull-up entre os pinos CH_PD e VCC da placa (mostrado na figura 4.9).

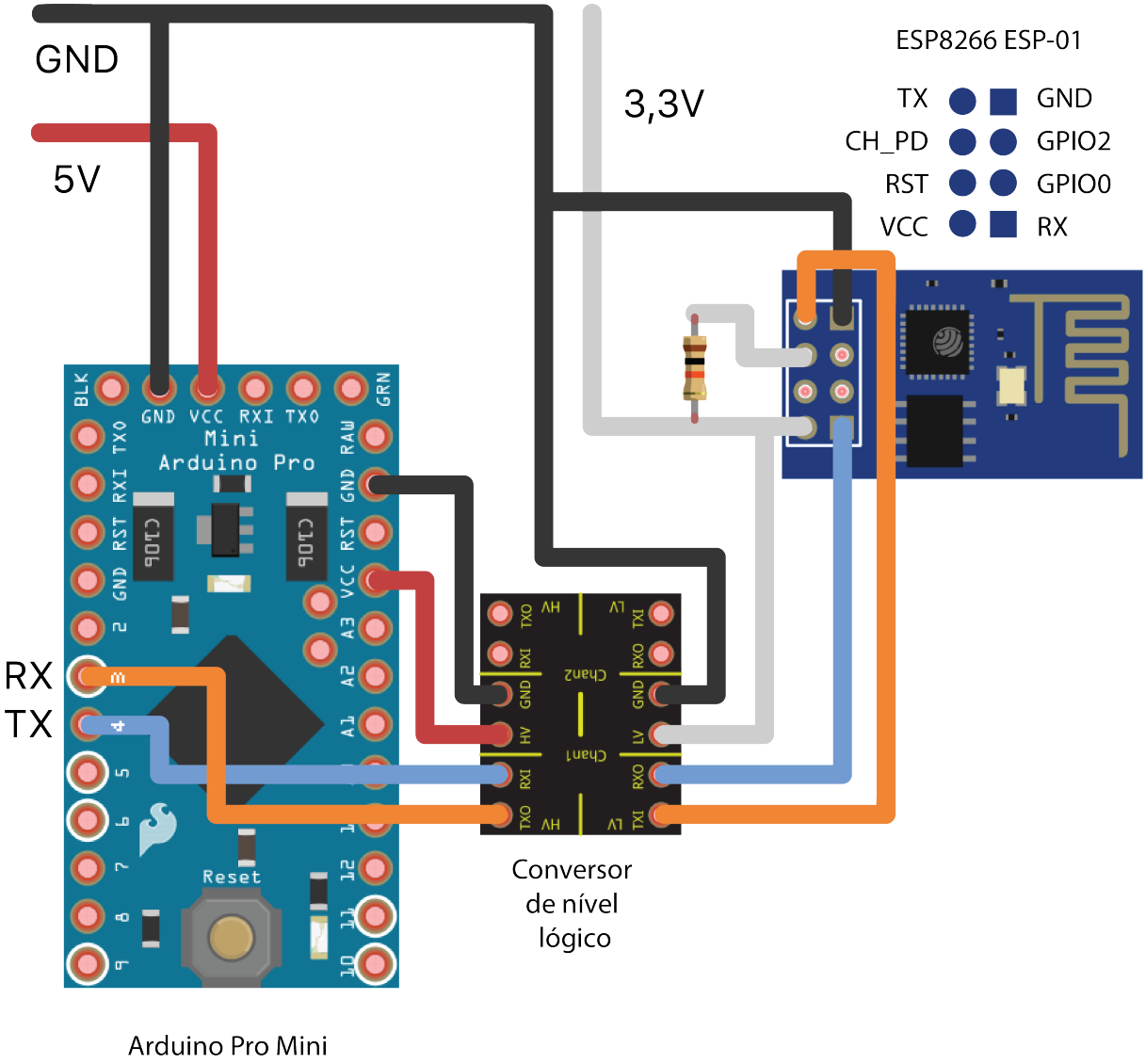


Figura 4.9 – Conexão dos pinos de transferência de dados e de alimentação entre as placas de WiFi (à esquerda) e Arduino (à direita). (Autor; As figuras dos componentes foram geradas pela ferramenta Fritzing - `http://fritzing.org/`)

Usa-se o conversor de nível lógico para reduzir a tensão dos pinos de transferência de dados, mantendo no máximo de 3,3V ao invés de 5V vindo diretamente da placa Arduino.

As ligações feitas já são suficientes para efetuar testes de transferência de dados entre as duas placas. O código que permite o teste básico de comunicação entre as placas está decrito no algoritmo 4.1:

Algoritmo 4.1 – Comunicação básica entre as placas Arduino e ESP8266. (Autor)

```
#include <SoftwareSerial.h>
```

```
//SoftwareSerial - Module(RX, TX);
SoftwareSerial ESP(3,4); //ESP8266 ESP-01

//BAUD RATE SETTINGS
#define SERIAL_BAUD_RATE 115200
#define ESP_BAUD_RATE 57600

void setup() {
 //Initialize Serial & ESP8266
 Serial.begin(SERIAL_BAUD_RATE);
 while (!Serial) {;} //Wait for Serial
 ESP.begin(ESP_BAUD_RATE);
 while (!ESP) {;} //Wait for ESP
}

void loop() {
 readSerial(); //Read from Serial to send to ESP
 readEsp(); //Read from ESP to send to Serial
}

void readEsp() {
 while(ESP.available()) { //When data is available
  String line = ESP.readStringUntil('\n'); //Get entire line
  Serial.println("ESP: " + line); //Show line
 }
}

void readSerial() {
 while (Serial.available()) { //When data is available
  String line = Serial.readStringUntil('\n'); //Entire line
  if (line.startsWith("ESP:")) { //Check if it starts with ESP
   String lineWithoutStart = line.substring(line.indexOf(":")+1);
   ESP.println(lineWithoutStart); //Show
  }
 }
}
```

O algoritmo 4.1 permite que por meio do monitor serial da aplicação Arduino IDE, seja possível enviar dados para a placa Arduino. A placa Arduino recebe as informações por comunicação serial e verifica se a linha começa com "ESP:" para então enviar para a placa ESP8266. As respostas enviadas pela placa ESP8266 são mostradas no monitor com o início "ESP:" em cada linha.

A janela do monitor serial pode ser aberta no menu 'Tools > Serial Monitor'. Usa-se esta janela para visualizar a comunicação entre as placas Arduino e WiFi. Após

carregar o código de comunicação básica entre as placas (no menu Sketch > Upload), pode-se utilizar como teste o envio do comando ’AT+RST’ para a placa WiFi. Ao enviar, tem-se a reinicialização do módulo e uma descrição com variados caracteres ilegíveis com o horário ’21:50:58’ e a palavra ’ready’ no final da resposta.

4.2.4 Preparação da placa GPRS/GSM SIM900a Mini

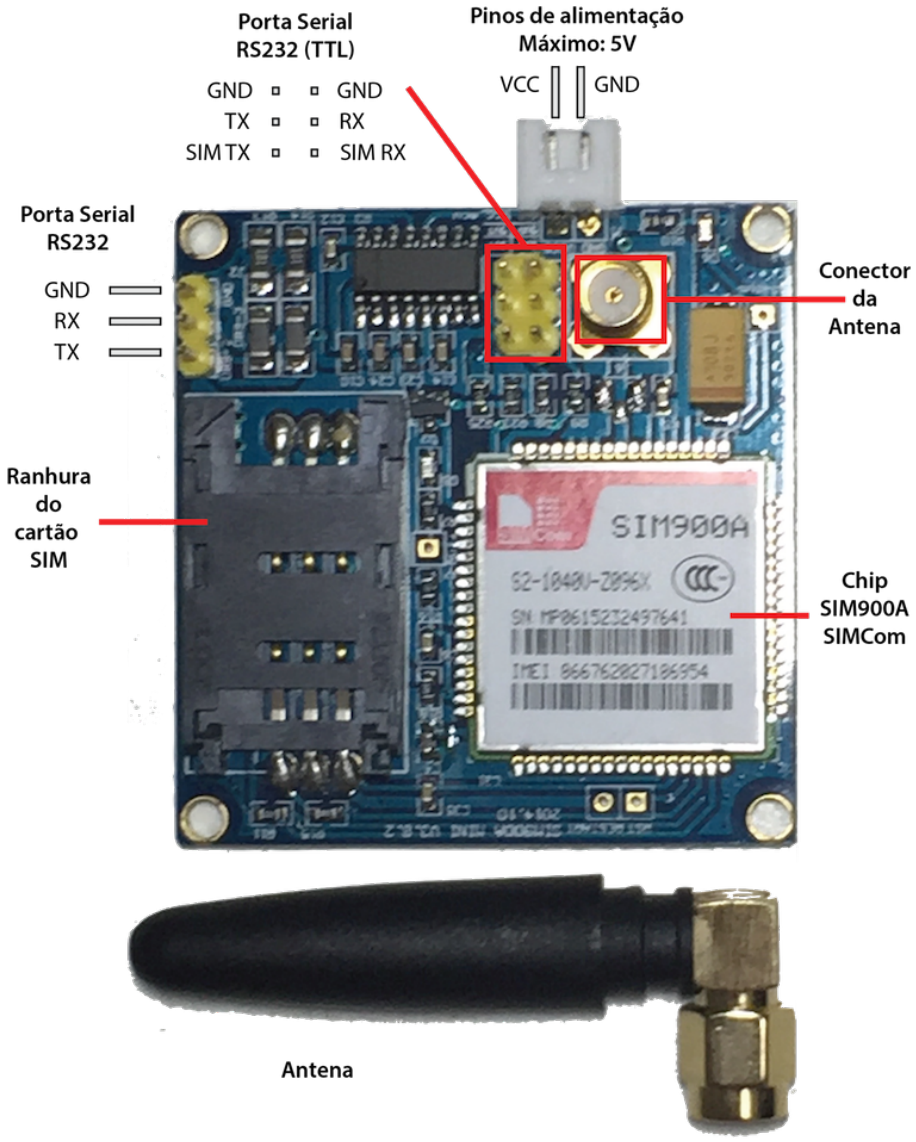


Figura 4.10 – Detalhes da placa do módulo SIM900a Mini. (Autor)

De acordo com a figura 4.10, a placa tem suporte para comunicação serial do tipo RS-232 e TTL. O modo apropriado para se utilizar com a placa Arduino seria o TTL,

já que não suporta comunicação diretamente com a porta RS-232, podendo danificar a placa.

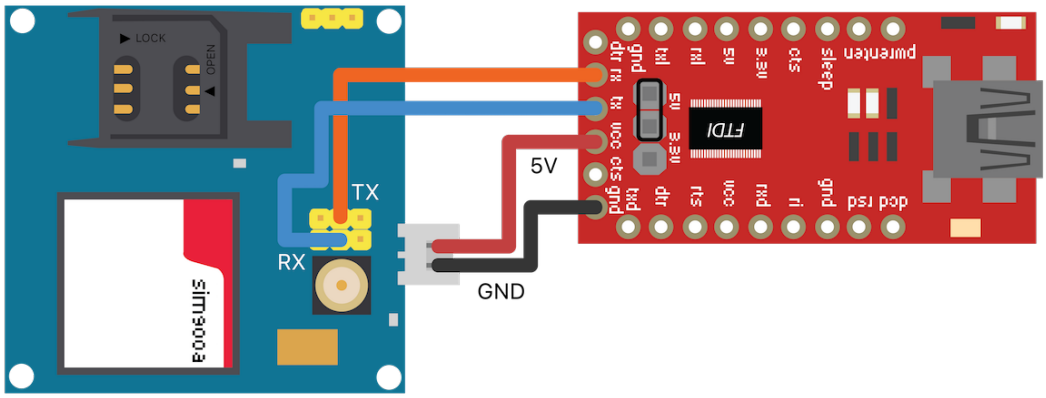


Figura 4.11 – Ligação entre o conversor serial e a placa SIM900a para atualização de firmware. (Autor)

Utilizando-se o modo TTL para realizar conexão com o conversor serial (placa FTDI FT232RL USB), efetua-se as seguintes conexões, que são mostradas na figura 4.11:

- Certifica-se de que o jumper de seleção de tensão encontra-se posicionado para o circuito da placa funcionar a 5V.
- Conecta-se o pino VCC da placa conversora serial com o pino VCC do módulo GPRS, como mostrado na figura 4.10.
- Liga-se os pinos RX e TX do módulo GPRS/GSM aos pinos TX e RX da placa conversora serial, respectivamente.
- Conecta-se os pinos GND das duas placas.

Para seu funcionamento, é necessário o uso de um firmware compatível com a rede regional. A versão do firmware compatível é a "1137B03SIM900M64_ST_ENHANCE" e pode ser encontrada no seguinte endereço: `http://dostmuhammad.com/sim900-firmware-update-tutorials-appnotes` (Acesso em 25/03/2016).

Após descarregar o arquivo "1137B03SIM900A64_ST_ENHANCE.cla" do endereço citado, é necessário o uso da ferramenta usada para carregar o firmware na placa.

A aplicação "Simcom - sim900 Customer flash loader V1.01" para o sistema operacional Windows, pode ser encontrada no seguinte endereço: `http://dostmuhammad.com/wp-content/uploads/Simcom_-_sim900_Customer_flash_loader_V1.01.rar` (Acesso em 25/03/2016). Os seguintes passos foram percorridos para o procedimento atualização do firmware:

1. Abrir a aplicação de atualização de firmware.
2. Informa-se a porta de comunicação serial que está disponível pela placa FTDI conectada ao USB.
3. Informa-se a taxa de transferência, que para a placa utilizada o valor selecionado foi 921600 bps.
4. Seleciona-se o arquivo do firmware em 'Download Settings', e então 'Browse...'.
5. Seleciona-se a opção 'START' para iniciar o carregamento do firmware.

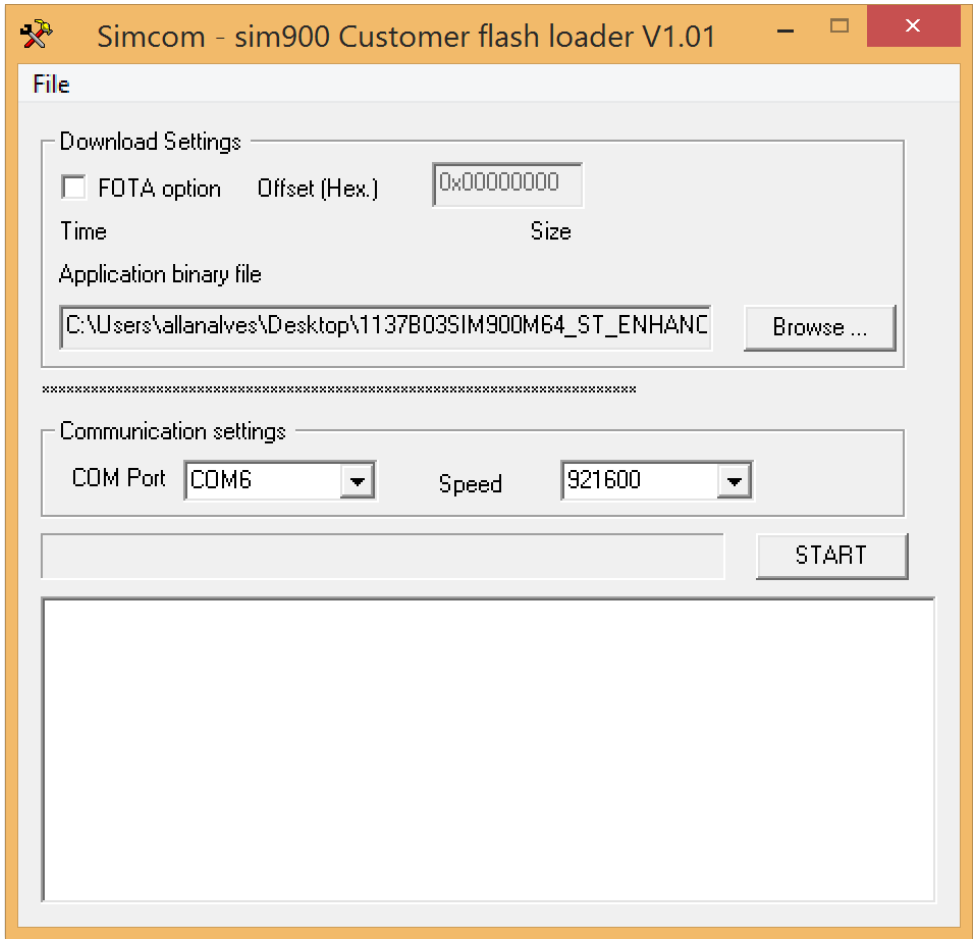


Figura 4.12 – Janela da ferramenta de atualização de firmware (Autor)

Após selecionadas as opções corretas, como mostra a figura 4.12, a seguinte mensagem é mostrada:

```
00’00’’000 - Waiting for board reset
00’01’’750 - Please power up the target.
```

Como descrito na mensagem, desligou-se e ligou-se o módulo rapidamente, provocando o efeito de reset, para então continuar o processo, que apresenta a seguinte mensagem:

```
00’00’’000 - Waiting for board reset
00’01’’750 - Please power up the target.
00’07’’640 - Target responding...
00’07’’640 - Downloading Flash Loader in RAM...
00’09’’875 - Flash Loader downloaded in RAM
```

Por fim, após quase 4 minutos (3’55”938), obteve-se uma mensagem de conclusão do carregamento do firmware, como mostrado na figura 4.13.

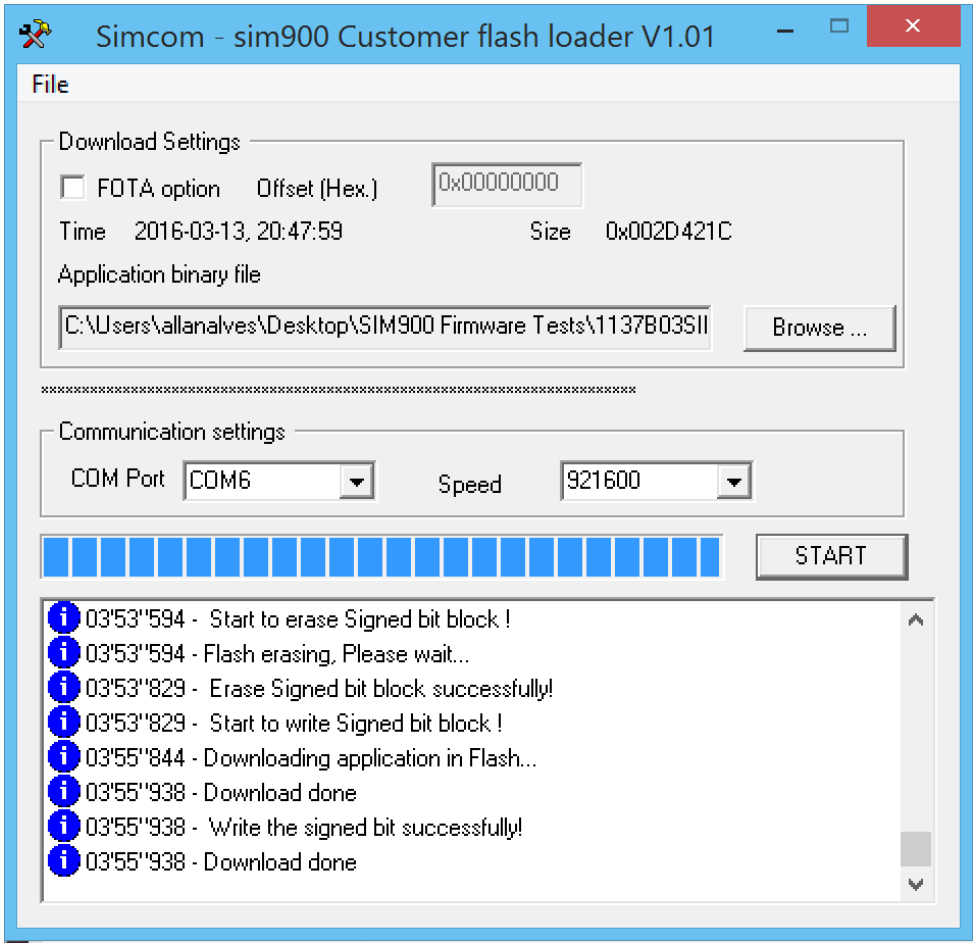


Figura 4.13 – Janela da ferramenta de atualização de firmware. Conclusão de carregamento. (Autor)

i

4.2.5 Definição da comunicação entre a placa GPRS/GSM SIM900a Mini e Arduino Pro Mini

As conexões feitas para a ligação entre as placas são mostradas na figura 4.14 e descritas em seguida:

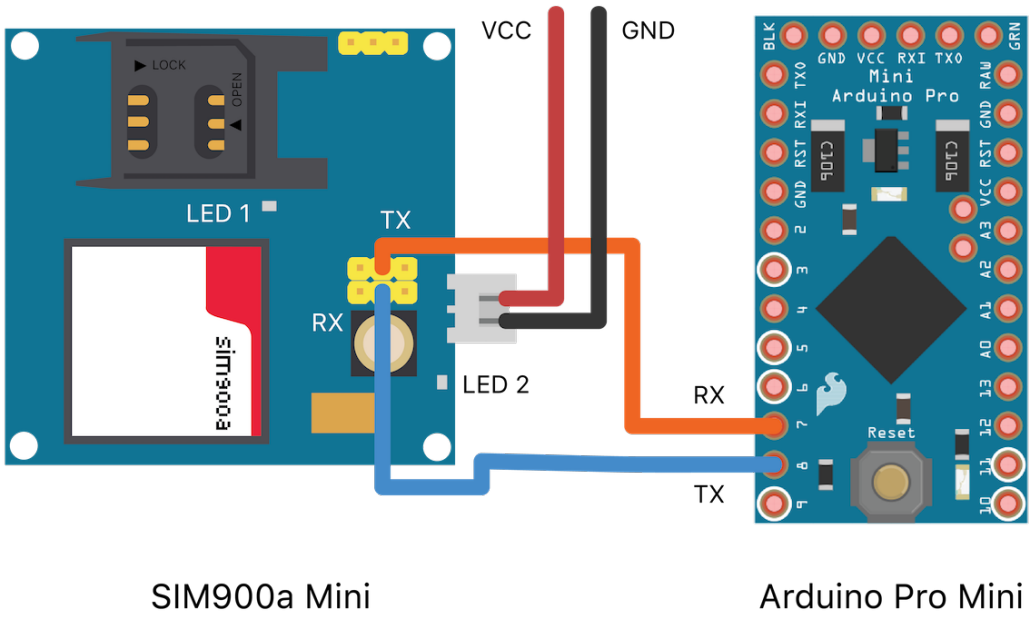


Figura 4.14 – Ligações de alimentação e comunicação entre as placas Arduino e SIM900A. (Autor; A figura da placa Arduino foi gerada pela ferramenta Fritzing - `http://fritzing.org/`)

- Certifica-se de que não há nenhum jumper na porta serial TTL. A retirada de jumpers define a opção de não utilizar RS-232.
- Liga-se o pino GND do módulo GPRS na mesma referência de terra do Arduino.
- Como a tensão de alimentação das duas placas é igual, liga-se o pino VCC da mesma fonte de tensão do Arduino.
- Conecta-se os pinos de transferência de dados RX e TX do módulo GPRS nos pinos 8 (TX) e 7 (RX) respectivamente.

Ambos LEDs 1 e 2 (cujas localizações são mostradas na figura 4.14), de cor vermelha, sinalizam o funcionamento da placa. O LED 1 é mantido aceso e o LED 2 pisca constantemente. Com as ligações feitas, pode-se prosseguir para realizar testes básicos de comunicação entre as placas.

Até o momento, verifica-se que o algoritmo usado prepara a placa Arduino para gerenciar as informações recebidas encaminhando-as, como no caso, envia dados para o módulo requisitado que foi indicado no início da linha enviada pelo monitor serial. Pode-se adicionar então o módulo SIM900a para receber comandos gerenciados pelo

Arduino, definindo assim a função da placa Arduino como uma central que comanda os módulos do sistema.

Com o algoritmo 4.2

Algoritmo 4.2 – Comunicação básica entre as placas Arduino, ESP8266 e SIM900A. (Autor)

```
#include <SoftwareSerial.h>

//SoftwareSerial - Module(RX, TX);
SoftwareSerial ESP(3,4); //ESP8266 ESP-01
SoftwareSerial GPRS(7,8); //SIM900A Mini

//BAUD RATE SETTINGS
#define SERIAL_BAUD_RATE 115200
#define ESP_BAUD_RATE 57600
#define GPRS_BAUD_RATE 57600

//MODUE IDENTIFICATION
#define ESP_MODULE_ID 100
#define GPRS_MODULE_ID 200

int moduleToListen;

void setup() {
  //Initialize Serial, ESP8266 & SIM900A
  Serial.begin(SERIAL_BAUD_RATE);
  while (!Serial) {;} //Wait for Serial
  ESP.begin(ESP_BAUD_RATE);
  while (!ESP) {;} //Wait for ESP
  GPRS.begin(GPRS_BAUD_RATE);
  while (!GPRS) {;} //Wait for GPRS
}

void loop() {
  //Read from serial & send data to modules
  readFromSerialWriteOnModules();
  if (moduleToListen == ESP_MODULE_ID) {
    readEsp(); //Listen only ESP
  } else if (moduleToListen == GPRS_MODULE_ID) {
    readGprs(); //Listen only GPRS
  }
}

void readEsp() {
  while(ESP.available()) { //When data is available
    String line = ESP.readStringUntil('\n'); //Get entire line
    Serial.println("ESP: " + line); //Show line
```

```
  }
}

void readGprs() {
  while(GPRS.available()) { //When data is available
    String line = GPRS.readStringUntil('\n'); //Get entire line
    Serial.println("GPRS: " + line); //Show line
  }
}

void readFromSerialWriteOnModules() {
  while (Serial.available()) {
    String line = Serial.readStringUntil('\n');
    if (line.startsWith("GPRS:")) { //Check if it starts with GPRS
      GPRS.listen(); //Start listening to GPRS
      moduleToListen = GPRS_MODULE_ID;
      String lineWithoutStart = line.substring(line.indexOf(":")+1);
      GPRS.println(lineWithoutStart); //Show entire line
    } else if (line.startsWith("ESP:")) { //Check if it starts with ESP
      ESP.listen(); //Start listening to ESP
      moduleToListen = ESP_MODULE_ID;
      String lineWithoutStart = line.substring(line.indexOf(":")+1);
      ESP.println(lineWithoutStart); //Show entire line
    }
  }
}
```

Em comparação com o algoritmo 4.1, adicionou-se suporte para a comunicação com os dois módulos. Criou-se identificação para cada módulo. Quando um módulo é citado no início da linha enviada pelo monitor serial (ESP: ou GPRS:), a placa Arduino lê somente o que este módulo envia. A figura 4.15 mostra um exemplo de envio do comando "AT" para cada um dos módulos, mostrando também as correspondentes respostas.

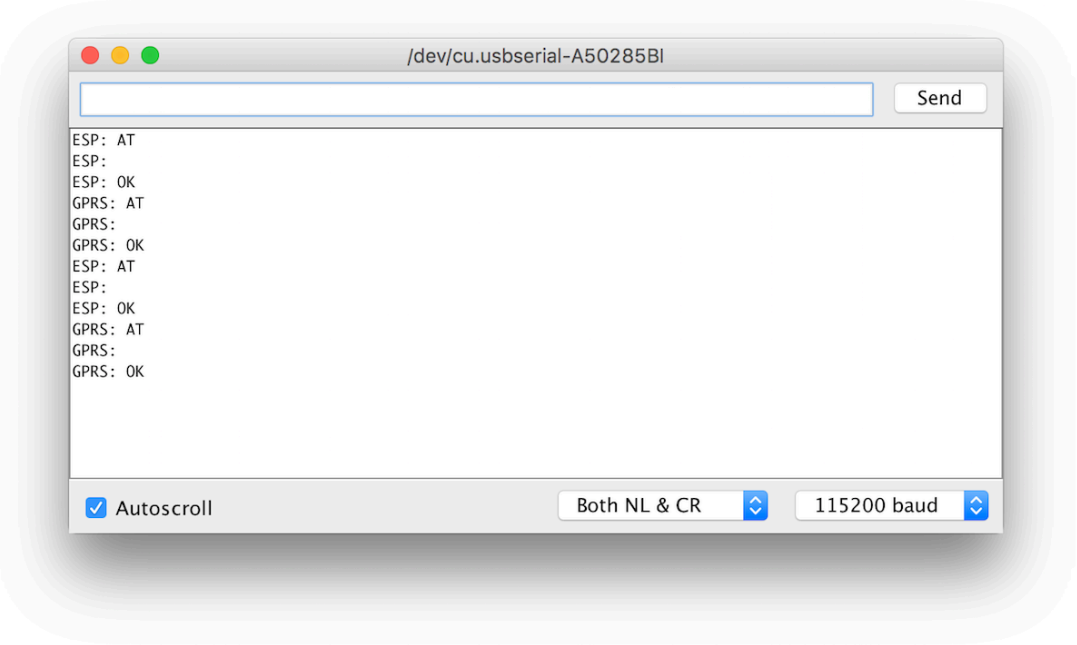


Figura 4.15 – Envio de comando AT para cada módulo conectado ao Arduino. (Autor)

4.2.6 Implementação de detecção de dispositivos autorizados conectados ao módulo WiFi

Com base no código criado anteriormente para gerenciar a comunicação entre os módulos e a placa Arduino, adiciona-se a função de ponto de acesso para o módulo WiFi, para permitir que dispositivos se conectem utilizando uma senha de acesso, definida juntamente com o nome do ponto de acesso. Para fins de testes, escolhe-se endereços MAC de dois dispositivos para definir em uma variável (’authMacAddresses’). No caso foram adicionados endereços de dois dispositivos móveis com sistemas operacionais diferentes. Os endereços podem ser encontrados nas configurações de cada um como mostrado na figura 4.3. O código escrito relacionado a estes procedimentos citados, é mostrado no algoritmo 4.3

Algoritmo 4.3 – Variáveis contendo o nome do ponto de acesso (SSID), senha e os endereços MAC dos dois dispositivos. (Autor)

```
//ESP SETTINGS
#define ESP_CHECK_CLIENTS_INTERVAL 1000 //Clients check interval
String espSSID = "SSV-AP"; //Access Point Name
String espPassword = "20973560"; //Access Point Password
String authMacAddresses = "8c:2d:aa:60:da:ad,60:be:b5:ea:52:a1";
unsigned long lastClientsCheckMillis; //Last clients check timestamp
```

```
boolean authStatusBefore = false;
boolean authStatus = false;
```

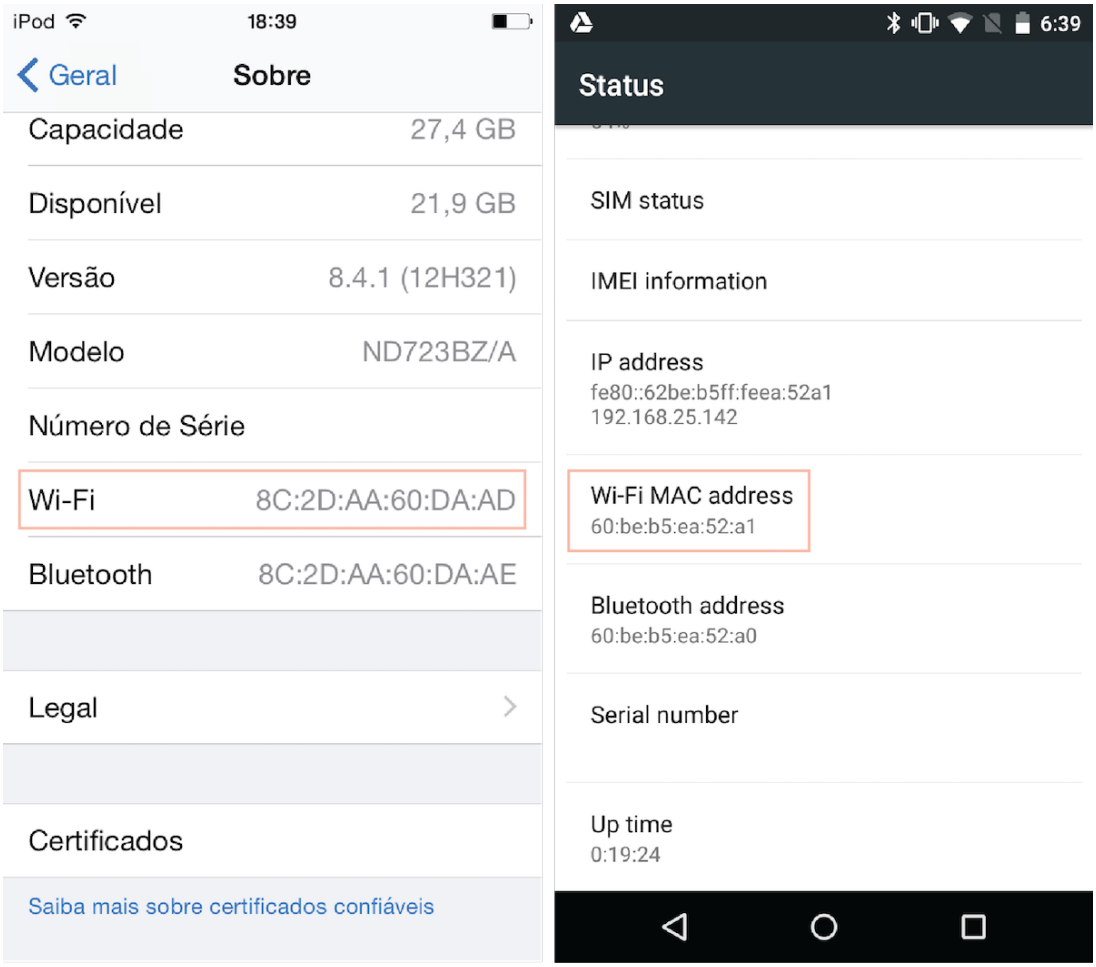


Figura 4.16 – Telas de configurações indicando localização dos endereços MAC (WiFi) dos dois dispositivos usados com função de dispositivos autorizados no sistema. (Autor)

As variáveis 'authStatusBefore' e 'authStatus', guardam os valores booleanos para controle de situação relativa à conexão de dispositivos autorizados. Para a criação de um temporizador que verifica os dispositivos conectados no módulo WiFi, usa-se uma variável para guardar o valor em milissegundos (tempo em que a placa Arduino está em funcionamento desde que foi ligado) da última vez em que ocorreu uma verificação. O algoritmo 4.4 mostra as novas funções e novos trechos adicionados:

Algoritmo 4.4 – Novas funções e trechos adicionados para executar a funcionalidade de detecção de dispositivos autorizados. (Autor)

```
void setup() {
  //Initialize Serial, ESP8266 & SIM900A
```

```
  Serial.begin(SERIAL_BAUD_RATE);
  while (!Serial) {;} //Wait for Serial
  ESP.begin(ESP_BAUD_RATE);
  while (!ESP) {;} //Wait for ESP
  GPRS.begin(GPRS_BAUD_RATE);
  while (!GPRS) {;} //Wait for GPRS

  moduleToListen = ESP_MODULE_ID; //Listen only ESP
  initEspModule(); //Setup ESP
}

void loop() {
  //Check devices connected to ESP
  espCheckClients();
  //Read from serial & send data to modules
  readFromSerialWriteOnModules();
  if (moduleToListen == ESP_MODULE_ID) {
    readEsp(); //Listen only ESP
  } else if (moduleToListen == GPRS_MODULE_ID) {
    readGprs(); //Listen only GPRS
  }
}

void initEspModule() {
  /*
    These initialization commands setup any ESP8266 module
    to work in AP mode and update the SSID and password
    if needed
  */
  //Set WiFi mode as access point mode
  ESP.println("AT+CWMODE=2");
  delay(BASE_DELAY);
  //Update WiFi SSID and Password
  ESP.println("AT+CWSAP=\"" + espSSID + "\"," + "\"" + espPassword + "\",9,2");
  delay(BASE_DELAY);
  //Multiple Connections
  ESP.println("AT+CIPMUX=1");
  delay(BASE_DELAY);
  //Create Server - Port 80
  ESP.println("AT+CIPSERVER=1,80");
  delay(BASE_DELAY);
  //Two Seconds Timeout
  ESP.println("AT+CIPSTO=1");
  delay(BASE_DELAY);
}

void espCheckClients() {
  if ((millis() - lastClientsCheckMillis) >= ESP_CHECK_CLIENTS_INTERVAL) {
    Serial.println("Checking Connected Devices...");
```

```
   ESP.listen(); //Listen to ESP
   lastClientsCheckMillis = millis(); //Store timestamp of the moment
   //Check Clients Connected to AP
   ESP.println("AT+CWLIF");
  }
}

void readEsp() {
  ESP.listen();
  while(ESP.available()) {
   String line = ESP.readStringUntil('\n');
   Serial.println("ESP: " + line);
   //Check any command after start of line
   String command = line.substring(line.indexOf("ESP: ")+1);
   //Check if received client information like IP and MAC Addresses
   if (command.startsWith("AT+CWLIF") || command.startsWith("192.")) { //Had response of
       CWLIF command
     authStatus = false;
     int indexOfMac = command.indexOf(",")+1;
     String mac = command.substring(indexOfMac,indexOfMac+17); //Avoid line break
     if (mac.length() >= 17) {
       int indexOfFoundAuth = authMacAddresses.indexOf(mac); //Try to found
       if (indexOfFoundAuth > -1) { //If Found
         Serial.println("Authorized MAC: " + mac);
         authStatus = true;
       }
     }
   }
  }
}
```

A função 'initEspModule()' define as configurações de acesso como SSID e senha do ponto de acesso e configura a placa para receber informações do aplicativo por comunicação TCP/IP. A função 'espCheckClients()' verifica a cada segundo os endereços IP e MAC dos dispositivos conectados no módulo WiFi. Quando carregado e executado, pode-se observar o seguinte conteúdo (a cada dois segundos) no monitor serial quando há um dispositivo conectado e este não está na lista de autorizados:

```
Checking Connected Devices...
ESP: AT+CWLIF
ESP: 192.168.4.2,a0:99:9b:2d:c9:6a
ESP:
ESP: OK
```

Quando conecta-se três dispositivos, sendo que destes, dois são autorizados,

pode-se observar o seguinte conteúdo:

```
Checking Connected Devices...
ESP: AT+CWLIF
ESP: 192.168.4.2,a0:99:9b:2d:c9:6a
ESP: 192.168.4.2,8c:2d:aa:60:da:ad
Authorized MAC: 8c:2d:aa:60:da:ad
ESP: 192.168.4.3,60:be:b5:ea:52:a1
Authorized MAC: 60:be:b5:ea:52:a1
ESP:
```

Utiliza-se a lista de dispositivos conectados ao módulo para analisar seus endereços MAC e IP, detectando então quais dentre os registrados são autorizados.

4.2.7 Preparação e integração do controle de travas automáticas

Os componentes da placa do controle de alarme e travas, foram identificados e são mostrados na figura 4.17

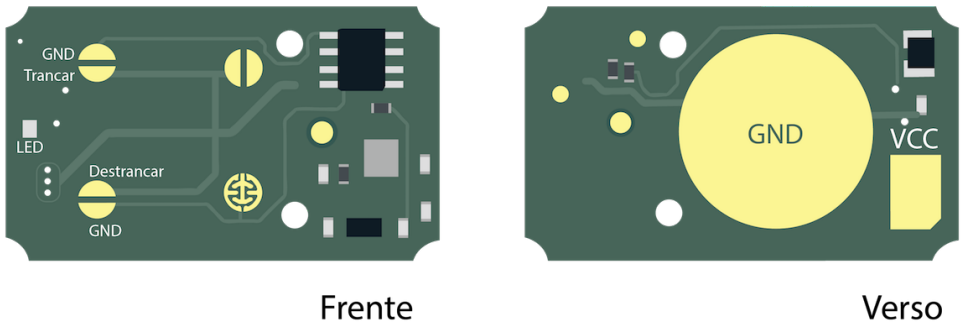


Figura 4.17 – Partes identificadas da placa do controle de alarme utilizado no projeto. (Autor)

A integração da placa do controle e a placa Arduino utiliza alguns componentes para complementar tanto na parte de alimentação como na lógica de ativação dos comandos de trancar e destrancar. Os seguintes procedimentos foram realizados para a montagem das conexões mostradas na figura 4.18:

- Solda-se fios para conexões nos contatos identificados cujas funções ativam os comandos de trancar e destrancar.

- Solda-se um fio ao contato de VCC da placa e outro para fazer ligação com a bateria, contectando-os respectivamente no contato coletor e no emissor de um transistor PNP.
- Conectar a base do transistor no pino 5 da placa Arduino para enviar sinal de definição se a placa de controle de alarme deve funcionar ou não.
- Usa-se uma bateria de 3V para aliemntar o controle de alarme, com seu contato GND em contato com o GND do controle de alarme.
- Utiliza-se dois transistores PNP para realizar o controle lógico dos comandos da placa do controle. Ao conectar o contato GND e o contato de trancar/destrancar, a função correspondente é ativada.
- Conecta-se as bases dos transistores nos pinos 10 e 11 da placa Arduino.

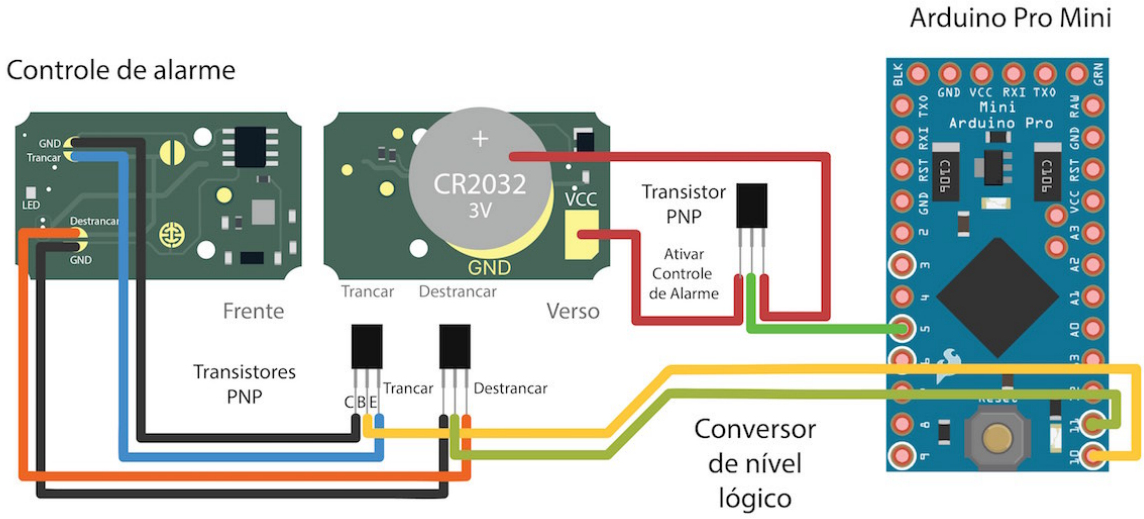


Figura 4.18 – Conexões para integração das placas do controle de alarme e Arduino. (Autor; A figura da placa Arduino foi gerada pela ferramenta Fritzing - `http://fritzing.org/`)

O algoritmo 4.5 mostra os trechos de código necessários para integrar a placa de controle ao sistema. A função 'loop()' chama as funções para trancar ('lockVehicle()') e destrancar ('unlockVehicle()') em intervalos de dois segundos. Posteriormente estas funções serão implementadas para serem utilizadas em momentos apropriados. O

controle de funcionamento da placa de controle de alarme é feito pelo pino de número 5, Ativando-o na função 'setup()', que é executada ao inicializar a placa Arduino.

Algoritmo 4.5 – Trechos de código para implementação da integração do controle de alarme. (Autor)

```
//Pins Variables
#define ALARM_VCC_PIN 5
#define ALARM_UNLOCK_PIN 10
#define ALARM_LOCK_PIN 11

//SERIAL SETTINGS
#define SERIAL_BAUD_RATE 115200

void setup() {
  //Pins
  pinMode(ALARM_VCC_PIN, OUTPUT);
  pinMode(ALARM_UNLOCK_PIN, OUTPUT);
  pinMode(ALARM_LOCK_PIN, OUTPUT);

  digitalWrite(ALARM_VCC_PIN, HIGH);
  digitalWrite(ALARM_UNLOCK_PIN, HIGH);
  digitalWrite(ALARM_LOCK_PIN, HIGH);
}

void loop() {
  //Alarm Commands Demonstration
  delay(2000);
  unlockVehicle();
  delay(2000);
  lockVehicle();
}

void lockVehicle () {
  //Alarm Control - Lock Command
  digitalWrite(ALARM_LOCK_PIN, LOW);
  delay(300);
  digitalWrite(ALARM_LOCK_PIN, HIGH);
}

void unlockVehicle () {
  //Alarm Control - Unlock Command
  digitalWrite(ALARM_UNLOCK_PIN, LOW);
  delay(300);
  digitalWrite(ALARM_UNLOCK_PIN, HIGH);
}
```

Para efetuar testes ao implementar, utiliza-se um 'buzzer' (dispositivo de sinaliza-

ção por áudio) quando o sistema efetua alguma ação específica. Em alguns trechos do código geral (apêndice B), pode-se observar seu uso em ações como acionamento do controle de alarme, requisições ao serviço em nuvem e detecção de movimento. É mostrada também como foi realizada sua montagem física no apêndice A.

O acionamento ao controle de alarme é utilizado ao detectar um dispositivo autorizado, como implementado anteriormente. Uma junção das lógicas de detecção de dispositivo autorizado e acionamento do controle de alarme é feita, gerando certas modificações na função 'checkClientsObserver()', como pode-se observar no código do apêndice B. O intuito desta implementação é fazer com que quando detectado um dispositivo autorizado, o sistema acione o destravamento do controle de alarme, permitindo acesso ao veículo. Ao detectar que não há mais nenhum dispositivo autorizado conectado, acionar o travamento do controle de alarme.

4.2.8 Implementação de recebimento de comandos enviados pelo aplicativo

Na função 'readEsp()' da implementação da placa Arduino, contém o trecho mostrado no algoritmo 4.6:

Algoritmo 4.6 – Detecção de comandos enviados pelo aplicativo. (Autor)

```
//Received Command from App
if (command.startsWith("+IPD") && authStatus) { //If Authorized
  if (ESP.find("comm")) {
    int commCode = ESP.parseInt();
    if (commCode == APP_COMMAND_TOGGLE_ALARM) { //Toggle Alarm Control
      if (lastIsLockedStatus) {
        unlockVehicle();
      } else {
        lockVehicle();
      }
    } else if (commCode == APP_COMMAND_REQUEST_SYNC) { //Update Authorized Devices
      findDevicesFromCloud();
    } else if (commCode == APP_COMMAND_REQUEST_LOCATION) { //Update Location
      getCurrentLocationAndSaveLastVehicleStatusOnCloud();
    }
  }
}
```

Ao receber dados do aplicativo conectado ao módulo WiFi, a placa ESP8266 envia os dados para a placa Arduino, que ao receber e detectar o trecho '+IPD' (que indica recebimento de dados por TCP/IP), procura pela chave 'comm' que contém em se-

guida um dos números de ações definidas no código. Suas reações ao recebimento destes códigos são implementadas na estrutura condicional descrita no algoritmo.

4.2.9 Preparação e integração do sensor de presença PIR

A ligação entre as placas é mostrada na figura 4.19, e é descrita nos seguintes passos:

- Alimenta-se a placa do sensor PIR com tensão de 5V pelo pino VCC.
- Conecta-se o lado do indicador da cifra 1 do resistor de 10 K Ohms na referência de alimentação.
- Conecta-se o lado do indicador de tolerância do resistor diretamente à saída do pino de saída do sinal da placa do sensor PIR.
- Liga-se a linha da saída do sinal do sensor diretamente no pino 2 da placa Arduino.
- Liga-se o pino GND do sensor na mesma referência da placa Arduino.

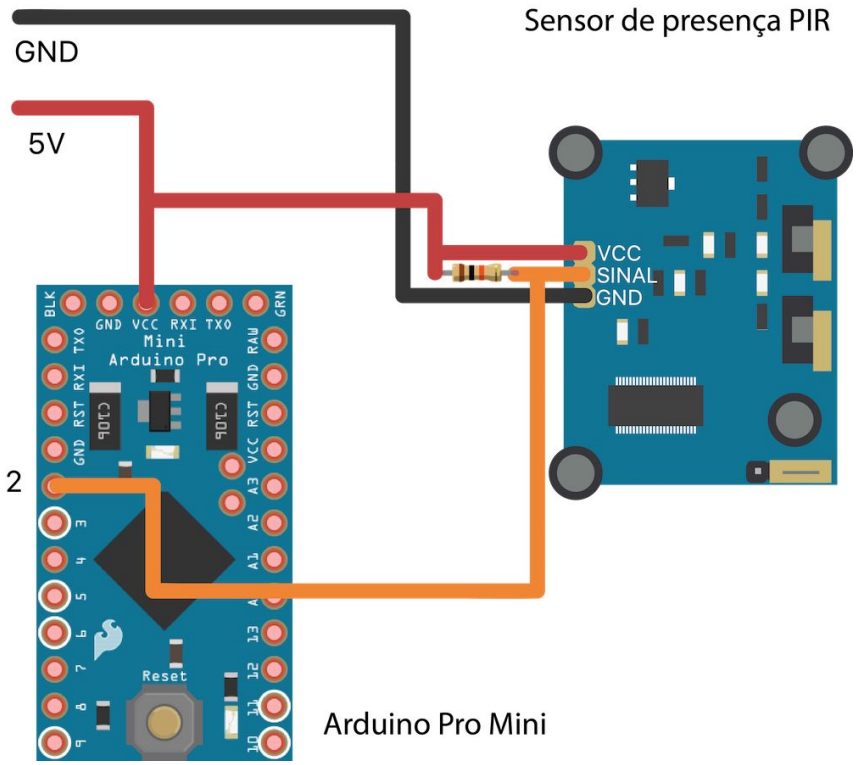


Figura 4.19 – Esquema de ligação entre as placas Arduino e do sensor PIR (Autor; A figura da placa Arduino foi gerada pela ferramenta Fritzing - `http://fritzing.org/`)

Na implementação da placa Arduino, detecta-se movimento apenas quando a variável que indica que o veículo está trancado está com valor positivo. Quando um movimento é detectado, a função 'getCurrentLocationAndSaveLastVehicleStatusOnCloud()' é chamada, identificando a localização atual do dispositivo e salvando-a no servidor juntamente com os dados de situação do veículo. A integração do sensor ao circuito pode ser observada no esquemático geral do apêndice A.

4.2.10 Implantação do dispositivo no interior de um veículo

Para a implantação do dispositivo em um veículo, é necessário regular a tensão de entrada do circuito e utilizar um meio de alimentar a placa com a bateria do veículo. Dois cabos originados do conjunto de fios utilizados para aparelhos de som automotivo e acessórios, foram extraídos para prover alimentação ao dispositivo. Utiliza-se um conector do tipo P4 (2,1 mm x 5,5 mm) unido por solda nos cabos extraídos e um conector do tipo J4 (2,1 mm x 5,5 mm) para ser utilizado na placa do circuito do dispositivo (sua montagem é mostrada no apêndice A). A figura 4.20 mostra um esquemático básico do uso dos conectores e do cabo de alimentação extraído do veículo.

Figura 4.20 – Esquemático básico de uso dos conectores e cabo de alimentação originado do veículo (Autor)

O regulador de tensão LM2596 mostrado na figura 4.21 foi utilizado para tornar possível a alimentação das placas que necessitam de 5V. Por meio da peça indicada na figura 4.21, ajusta-se a tensão de saída do regulador, sendo no caso 5V. O esquemático do apêndice A mostra o componente regulador integrado ao circuito.

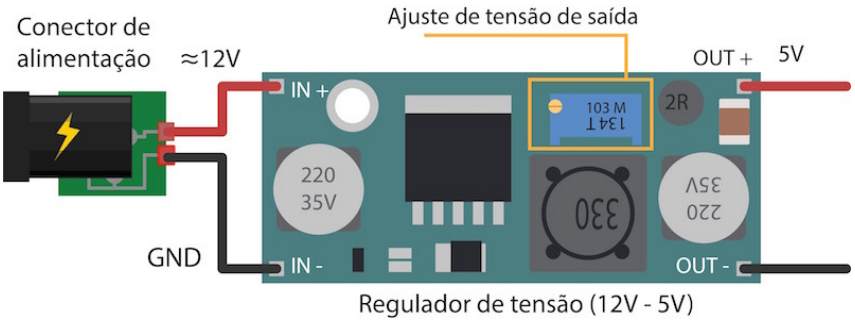


Figura 4.21 – Regulador de tensão LM2596. (Autor; A figura do conector de alimentação foi gerada pela ferramenta Fritzing - `http://fritzing.org/`)

Para utilização de tensão de aproximadamente 3,3V para alguns componentes como o módulo WiFi, usou-se o regulador de tensão LM1117T-3.3 (4.22) juntamente com dois capacitores de 10 Microfarads para alimentar o módulo WiFi e para servir de referência ao conversor de nível lógico. É possível verificar no esquemático geral do circuito do apêndice A, a forma como este regulador foi montado juntamente com os componentes citados anteriormente.

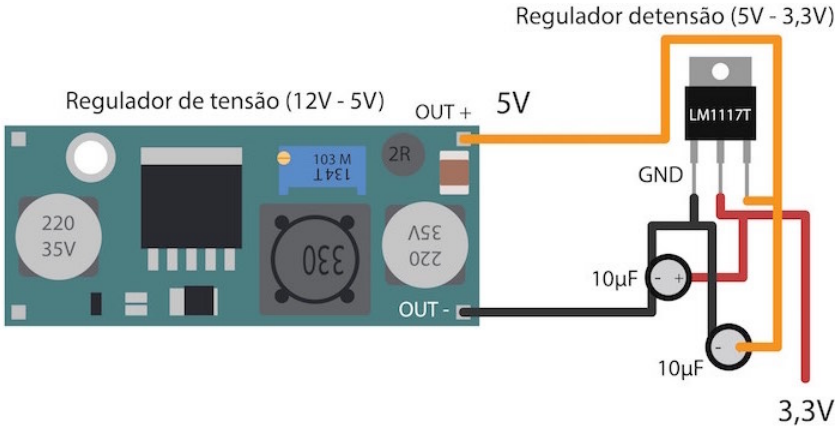


Figura 4.22 – Regulador de tensão LM1117T-3.3. (Autor)

Um botão de ligar ou desligar o circuito foi instalado ao lado do conector de alimentação. Pode-se observá-lo no apêndice A.

Neste ponto, conclui-se os procedimentos sobre a primeira etapa, que descreve a construção do equipamento e implementação do hardware. Em seguida nos itens 4.2.11 a 4.2.15, descreve-se sobre a implementação do serviço de comunicação e armazenamento de informações na nuvem, que seria a segunda etapa do desenvolvimento do projeto.

4.2.11 Preparação do serviço MBaaS para armazenamento e manipulação de dados

Com uma conta criada gratuitamente no site `http://www.parse.com/`, pode-se criar, por meio da interface de gerenciamento do sistema do serviço, um aplicativo reservado para o projeto. Selecionando a opção 'Create a new app' logo na página 'Dashboard', abre-se uma janela (mostrada na figura 4.23) para informar o nome do aplicativo e o tipo de banco de dados. O tipo de banco escolhido foi o 'Parse Data', que é voltado para ser manipulado pelos SDKs (Kits de desenvolvimento) providos pelo serviço, para prover mais praticidade ao implementar.

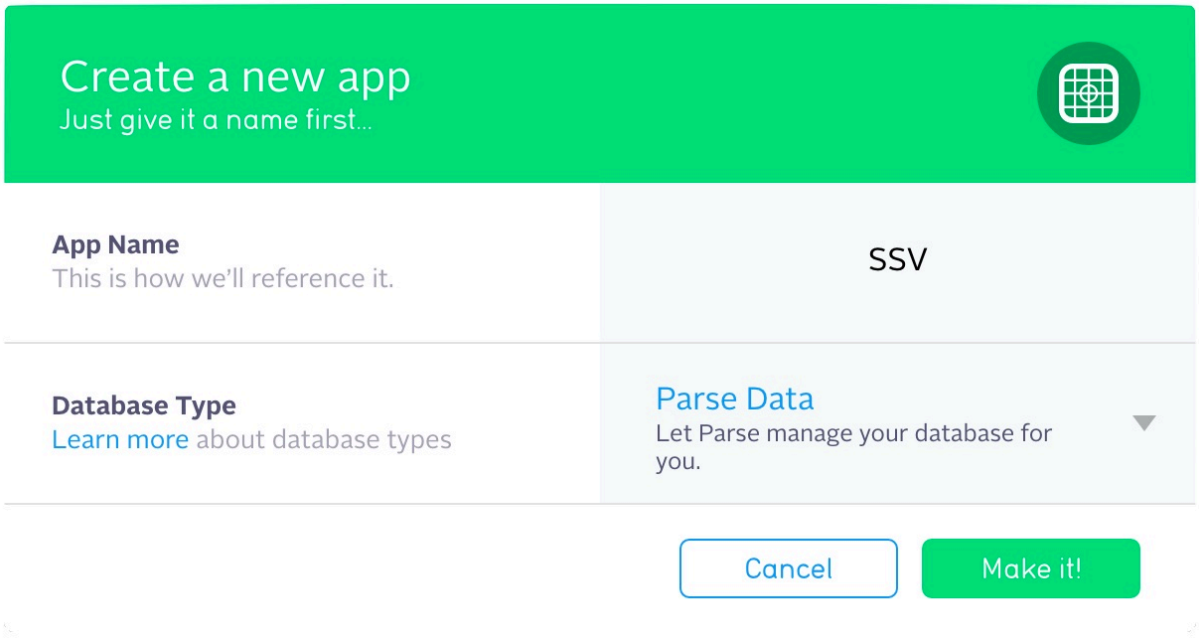


Figura 4.23 – Janela para criação de um novo aplicativo. (Disponível em `www.parse.com/apps/` - Acesso em 24/05/2016)

Na aba 'Core', onde se gerencia características gerais de um aplicativo, como armazenamento, execução de código e trabalhos automáticos pelo servidor, registro de eventos (log) e configurações, pode-se acessar a seção de armazenamento, entitulada como 'Data'. É possível criar classes de objetos que são a base do funcionamento de manipulação de dados em nuvem do sistema. Com base nas necessidades do sistema, as seguintes classes são utilizadas:

- Installation: instalações
- Device: dispositivos

- Event: eventos
- VehicleStatus: situação do veículo

A classe 'Installation', é uma das classes tratadas como comuns em aplicativos (dentre elas estão usuários, função/cargo e produto). Tem como função salvar informações relacionadas ao ato de instalar o aplicativo em um dispositivo, como: identificação do aplicativo, versão, nome, tipo de dispositivo, token do dispositivo e identificação de instalação. Como já existe ao criar um aplicativo, não é necessário criar.

O botão 'Add Class' permite criar uma classe para a modelagem do banco de dados do sistema.

A classe 'Device', ou dispositivo, é criada com o tipo 'Custom', ou personalizada. É utilizada para salvar informações sobre os dispositivos autorizados. As colunas, ou propriedades criadas para a classe são as seguintes:

- 'objectId' (string): propriedade criada automaticamente para identificar o dispositivo.
- 'macAddress' (string): endereço MAC do dispositivo.
- 'deviceModel' (string): nome do modelo do dispositivo.
- 'deviceName' (string): nome do dispositivo configurado pelo seu usuário.
- 'createdAt' (data): data da inclusão do objeto. Propriedade criada automaticamente.
- 'updatedAt' (data): data da última modificação feita no objeto. Propriedade criada automaticamente.
- 'ACL' (ACL): tipo de permissão atribuída a manipulação do objeto. Propriedade criada automaticamente.

A classe 'Event', ou evento, é criada com o intuito de salvar avisos para envio aos usuários do sistema de segurança veicular. Nela, além das propriedades criadas automaticamente (como citadas no detalhamento da classe 'Device'), guarda um número do evento para identificação, descrição para ser mostrada na aplicação do usuário e descrição para a notificação que será recebida pelos usuários. Foram criados somente

quatro objetos na tabela desta classe, que não são modificados ao longo do funcionamento do sistema. A página de gerenciamento dos objetos desta classeé mostrada na figura 4.24.

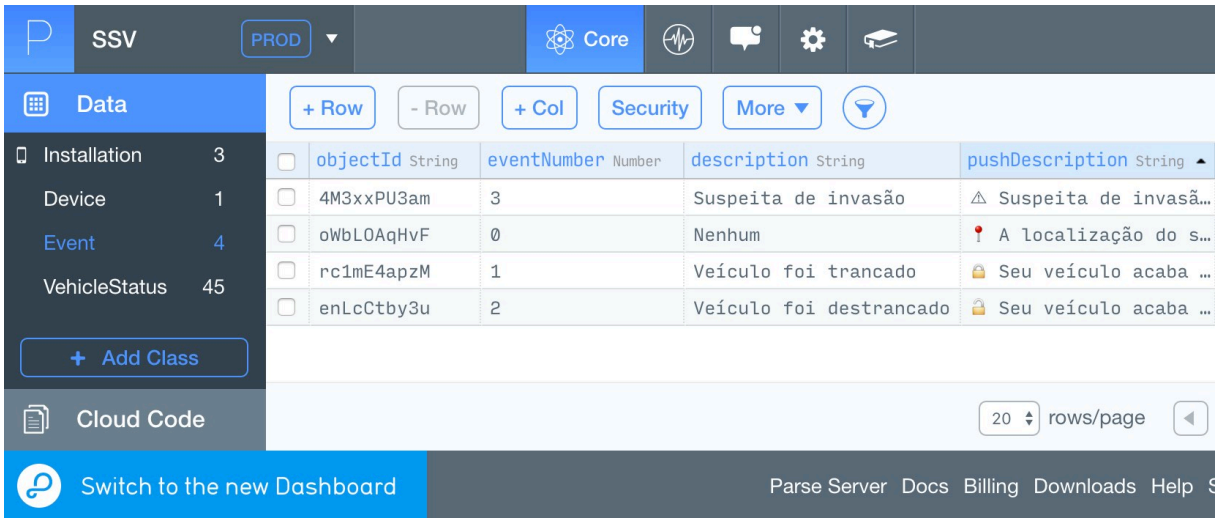


Figura 4.24 – Página para gerenciamento da classe 'Event'. (Disponível em `www.parse.com/apps/` - Acesso em 24/05/2016)

A classe 'VehicleStatus', é criada para salvar as principais informações que o sistema armazena. As propriedades criadas para esta classe são:

- 'isLocked' (booleano): indica se o veículo está trancado ou não no momento do envio do evento á nuvem.
- 'latitude' (string) e 'longitude' (string): coordenadas geográficas identificadas pelo módulo GPRS utilizado no sistema.
- 'event' (número): número de identificação de um dos eventos que populam a tabela 'Event'. Corresponde ao evento ocorrido no momento em que a situação do veículo é enviada á nuvem.

Ao criar este modelo de dados, já se torna possível a criação, modificação e exclusão destes objetos. Somente objetos da classe 'Device' serão manipulados pelo usuário da aplicação móvel. Os objetos da classe 'VehicleStatus' são adicionados constantemente e os objetos da classe 'Event' não são modificados nem excluídos, sendo assim estáticos.

4.2.12 Preparação para criação das rotas para realização de requisições externas

Na página 'Settings', ou configurações, pode-se configurar características e o funcionamento de mecanismos específicos do serviço, voltado para o aplicativo do sistema. Na seção 'Hosting', escolhe-se um caminho para que seja criada uma página na web reservada para o aplicativo, disponibilizando uma URL para que o dispositivo físico se conecte para efetuar requisições HTTP.

Como mostra a figura 4.25, basta utilizar um nome para a geração da página pela URL.

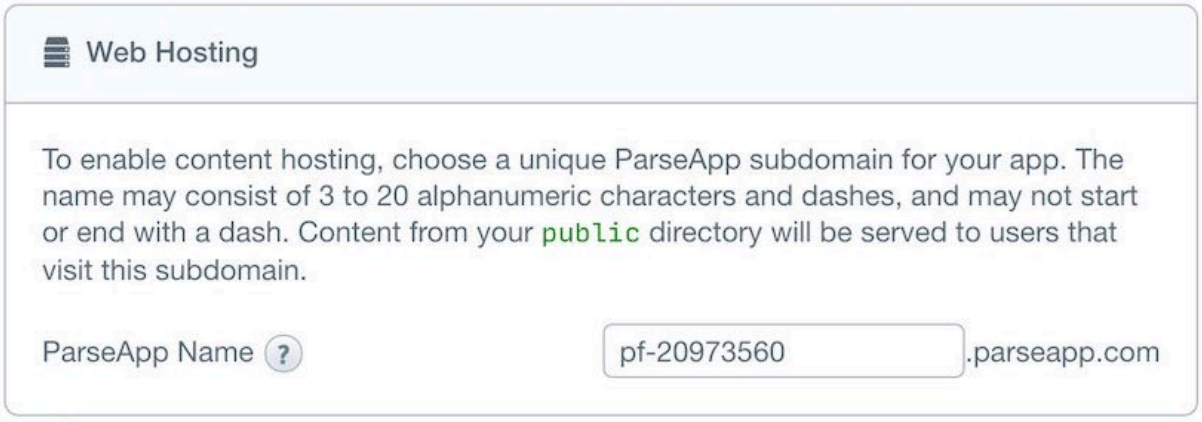


Figura 4.25 – Página de configurações de hospedagem, para disponibilizar um canal de acesso (URL) e prover recursos para o aplicativo. (Disponível em `www.parse.com/apps/` - Acesso em 24/05/2016)

Ao acessar por um navegador, pode-se visualizar a página, como mostra a figura 4.26.

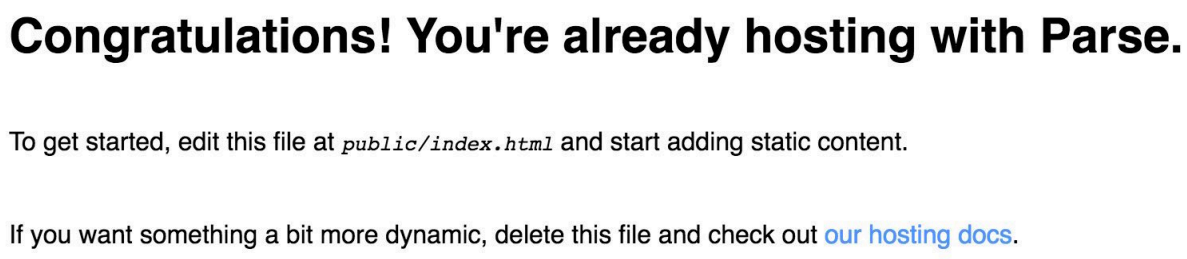


Figura 4.26 – Página gerada pela ativação do serviço de hospedagem. (Autor)

Novamente na aba 'Core', pode-se acessar a funcionalidade chamada 'Cloud Code', onde pode-se visualizar os arquivos de código, que são utilizados nas chamadas ao

serviço criado. Para instalar a ferramenta a interagir com os arquivos de código e de páginas web, deve-se utilizar o terminal de um ambiente Linux/Unix ou Mac OS X. Ao abrí-lo, executa-se a seguinte linha:

```
curl -s https://www.parse.com/downloads/cloud_code/installer.sh | sudo /bin/bash
```

Este comando instala a ferramenta chamada 'parse' no diretório '/usr/local/bin/-parse'. Carrega-se os arquivos para serem editados localmente (para posterior upload) utilizando os comandos (seguidos de suas respostas) em seguida:

```
$ parse new

Would you like to create a new app, or add Cloud Code to an existing app?
Type "(n)ew" or "(e)xisting":
```

Escolhe-se a opção de carregar um aplicativo existente, digitando a letra 'e'. A lista de aplicativos é listada e deve-se digitar o número correspondente ao aplicativo do sistema. Após escolher, a seguinte mensagem com opções é mostrada:

```
Which of these providers would you like use for running your server code:
1) Heroku (https://www.heroku.com)
2) Parse (https://parse.com/docs/cloudcode/guide)
```

Escolhe-se a opção '2', 'Parse', e em seguida, espera-se a seguinte mensagem como resposta:

```
Please enter the name of the folder where we can download the latest deployed
Cloud Code for your app "SSV"

Directory Name:
```

É necessário digitar o nome de um diretório para que os arquivos sejam carregados. Em seguida uma nova mensagem é mostrada. Para optar por carregar os arquivos já existentes, basta seguir sem digitar nada.

```
You can either set up a blank project or download the current deployed Cloud Code.
Please type "(b)lank" if you wish to setup a blank project, otherwise press ENTER:
```

Como resposta de sucesso, a seguinte mensagem é mostrada:

```
Successfully downloaded Cloud Code to "/Users/allanalves/ssv".
Successfully configured email for current project to: "allanalves90@gmail.com"
```

Acessando o subdiretório 'cloud', nota-se a existência do arquivo (na linguagem Ja-

vaScript) 'main.js', que se trata do arquivo principal, executado em primeiro lugar. Após realizar modificações neste diretório, sincroniza-se com o servidor executando a seguinte linha na raíz (em seguida a resposta de sucesso ao comando) do carregamento feito e indicado anteriormente (no caso '/Users/allanalves/ssv'):

```
$ parse deploy

Uploading source files
Uploading recent changes to scripts...
The following files will be uploaded:
...
Uploading recent changes to hosting...
The following files will be uploaded:
...
Finished uploading files
New release is named vx (using Parse JavaScript SDK vx.x.x)
```

4.2.13 Criação da rota para receber endereços de dispositivos autorizados

Cria-se um novo arquivo com o nome de 'app.js', que se reserva somente para executar a lógica das chamadas. É necessário incluí-lo em 'main.js', da seguinte forma:

```
require('cloud/app.js');
```

Os arquivos aqui citados ('app.js', 'main.js') em suas versões completas podem ser encontrados no apêndice C. No arquivo 'app.js', inclui-se as seguintes linhas de código:

```
express = require('express');
app = express();
var apiKey = '9Qv1SPg0W3jCz5bNlF7Ldny7w3yBYf1rIQlgvsr5';
app.use(express.bodyParser()); //Middleware for reading request body
```

A primeira e segunda linha são utilizadas para utilizar um framework que contém recursos para desenvolver aplicações web. A terceira linha define a chave da API para aceitar comunicação de requisições com o cliente, que seria no caso o dispositivo GPRS. Esta chave é gerada ao criar um aplicativo novo no Parse, sendo encontrada em 'Settings > Keys > Application ID'.

O algoritmo 4.7, mostra como é realizado o tratamento de uma requisição enviada para o endereço: `pf-20973560.parseapp.com/macDevices`

Algoritmo 4.7 – Função que retorna os endereços MAC dos dispositivos que estão no banco de dados. (Autor)

```
//To Receive MAC Addresses of All Authorized Devices
app.post('/macDevices', function(req, res) {
  var requestApiKey = req.body['x-key'];

  //Check the API key
  if (requestApiKey === undefined || requestApiKey !== apiKey) {
    res.status(401).send({error:"Missing or incorrect API-key."});
    return;
  };

  //Query
  var Device = Parse.Object.extend('Device');
  var query = new Parse.Query(Device);
  query.find({
   success: function(results) {
      //Retrieved Devices - Check Length:
    alert("Successfully retrieved " + results.length + " devices.");
      if (results.length) { //If there's any device
        var macs = [];
        // Do something with the returned Parse.Object values
        for (var i = 0; i < results.length; i++) {
          var object = results[i];
          var macAddress = object.get('macAddress');
          macs.push(macAddress);
        }
        //[DM - Devices MAC Addresses Start
        //DM] - Devices MAC Addresses End
        var macsString = macs.toString();
        macsString = macsString.split(":").join("");
        res.send('[DM' + macsString + 'DM]');
        sendPush("Sistema inicializado. " +
        "A lista de dispositivos foi sincronizada.", 0);
      } else { //List has no devices
        sendPush("Nao ha nenhum dispositivo autorizado. " +
        "Adicione um dispositivo.", 0);
        res.send({r: '[DM-DM]'});
      }
   },
   error: function(error) {
    alert("Error: " + error.code + " " + error.message);
   }
  });
});
```

Na primeira parte do algoritmo, valida-se a chave da API ('requestApiKey'), para

prosseguir. No trecho em que se declara a variável 'query' ('var query'), é realizada uma requisição que busca todos os dispositivos autorizados (máximo dois) da classe 'Device', utilizando todos os objetos recebidos para extrair seus endereços MAC da propriedade 'macAddress', e assim, retornar para uma sequência de caracteres contendo o endereço MAC de dois dispositivos.

4.2.14 Criação da rota de envio de informações sobre a situação do veículo

No arquivo 'app.js', cria-se outra função que armazena informações da situação do veículo. A função é mostrada no algoritmo 4.8.

Algoritmo 4.8 – Função que recebe dados da situação do veículo e retorna resposta booleana de sucesso ou erro. (Autor)

```
//To Send Vehicle Status
app.post('/vStatus', function(req, res) {
  var requestApiKey = req.body['x-key'];
  var isLocked = req.body.isLocked;
  var event = req.body.event;
  var latitude = req.body.lat;
  var longitude = req.body.lng;

  //Check the API key
  if (requestApiKey === undefined || requestApiKey !== apiKey) {
    res.status(401).send({error:"Missing or incorrect API-key."});
    return;
  };

  //Validate Locked Status
  if (isLocked === undefined) {
    res.status(400).send({error:"isLocked missing."});
    return;
  } else {
    isLocked = Boolean(isLocked);
    if (typeof isLocked !== "boolean") {
      res.status(400).send({error:"isLocked is not a boolean."});
      return;
    }
  };

  //Validate Moment Action
  if (event === undefined) {
    res.status(400).send({error:"Event missing."});
    return;
  } else if (typeof event !== "number") {
    res.status(400).send({error:"Event is not a number."});
```

```
    return;
  };

  //Validate Latitude
  if (latitude === undefined) {
    res.status(400).send({error:"Latitude missing."});
    return;
  } else if (latitude.length <= 0) {
    res.status(400).send({error:"Latitude sent without length."});
    return;
  } else if (typeof latitude !== "string") {
    res.status(400).send({error:"Latitude is not a valid string."});
    return;
  };

  //Validate Longitude
  if (longitude === undefined) {
    res.status(400).send({error:"Longitude missing."});
    return;
  } else if (longitude.length <= 0) {
    res.status(400).send({error:"Longitude sent without length."});
    return;
  } else if (typeof longitude !== "string") {
    res.status(400).send({error:"Longitude is not a valid string."});
    return;
  };

  //Create the Vehicle Status object.
  var VehicleStatus = Parse.Object.extend("VehicleStatus");
  var newVehicleStatus = new VehicleStatus();
  newVehicleStatus.set('isLocked', isLocked);
  newVehicleStatus.set('event', event);
  newVehicleStatus.set('latitude',latitude);
  newVehicleStatus.set('longitude', longitude);

  newVehicleStatus.save(null, {
    success: function (newVehicleStatus) {
      //Everything ok. report error to client.
      res.send({"r": "[DMOK]"});
    },
    error: function (error) {
      //Error saving the log. Report error to client.
      res.status(400).send({error:"Log save error: " +
                              error.code + " " +
                              error.message});
    }
  });
});
```

Em seu início, são criadas variáveis com o conteúdo das informações enviadas na requisição (objeto 'req' recebido pela função '.post' de 'app'), como chave da API, um valor booleano que indica se o veículo estava trancado ou não no momento, o número do tipo de evento, e as coordenadas geográficas. Em seguida, todas essas variáveis são validadas dentro de critérios básicos, como verificação de acordo com o tipo de conteúdo esperado ou se há conteúdo nestas variáveis. No trecho que a variável 'newVehicleStatus' é declarada, cria-se um novo objeto da classe "VehicleStatus", salvando então com a função '.save' deste objeto. Define-se por final o retorno que será enviado em 'success:', caso ocorra o processo como esperado, e 'error:', caso houver algum problema ao salvar.

O valor que representa o tipo do evento recebido é provindo de uma requisição enviada pelo dispositivo de segurança, que pode especificar a mensagem da notificação que será enviada para os smartphones e a descrição deste evento nos registros do aplicativo. Pode-se haver um valor que representa a inexistência de um evento no momento em casos frequentes de atualização de localização.

4.2.15 Implementação para identificar localização e enviar eventos para o serviço utilizando as rotas criadas

O dispositivo GPRS, provê funções de comunicação HTTP e identificação de coordenadas geográficas. Por meio de certos comandos AT, o módulo estabelece conexão com a provedora, e em seguida, inicializa conexão por HTTP. No apêndice B, encontra-se o algoritmo completo da implementação da placa Arduino e sua integração com os demais módulos. São incluidas também as funcionalidades de requisição HTTP e identificação de localização geográfica. A função 'initGprsModule()' faz uso do comando 'AT+SAPBR', que define configurações de portadora para aplicações baseadas em endereço IP, como no caso uma conexão GPRS:

```
"AT+SAPBR=3,1,"CONTYPE","GPRS"
"AT+SAPBR=3,1,"APN","tim.br"
"AT+SAPBR=1,1"
```

Os comandos citados acima definem, respectivamente o tipo de conexão, o ponto de acesso de conexão e abre a conexão com a portadora. O método 'initGprsModule()' executa estes comandos de maneira que, se algum falhar, por instabilidade de

conexão por exemplo, sabe-se qual etapa repetir e continuar, por serem comandos indispensáveis para o funcionamento do dispositivo. Enquanto o sistema executar os comandos desta etapa de inicialização, o LED de cor vermelha pisca ou se mantém aceso, indicando que o sistema está ocupado inicializando. O código implementado para seu funcionamento pode ser encontrado no apêndice B, e sua implementação física no apêndice A.

Para executar uma requisição HTTP, usa-se a função 'initializeHttp(String path)', que recebe como único parâmetro o endereço da chamada. Os comandos AT em seguida configuram o módulo GPRS para iniciar uma requisição:

```
"AT+HTTPINIT"
"AT+HTTPPARA="URL","(ENDERECO)"
"AT+HTTPPARA="CID",1"
"AT+HTTPPARA="CONTENT","application/json"
```

Os comandos citados acima definem respectivamente a inicialização do serviço HTTP, o endereço da requisição (na marcação '(ENDERECO)'), o parâmetro 'CID' para identificar a chamada e o parâmetro para definir que o conteúdo é do tipo JSON (formato usado para troca de dados).

A função 'submitRequestWithBody(String jsonBody)', que recebe como único parâmetro o conteúdo JSON, define os dados que serão enviados pela requisição. Os seguintes comandos AT são utilizados neste trecho:

```
"AT+HTTPDATA=(DADOS),6000"
"AT+HTTPACTION=1"
```

Os comandos citados acima definem respectivamente a inserção dos dados a serem enviados com o tempo máximo em milissegundos para a inserção dos dados e o método da ação HTTP, que seria sempre do tipo 'POST'.

O método 'terminateHttpService()', que é chamado após toda requisição feita, finaliza o serviço HTTP inicializado anteriormente executando o seguinte comando:

```
"AT+HTTPTERM"
```

A função 'readGprs()' é executada repetidamente sendo chamada pela função 'loop()', caso o módulo a ser 'ouvido' pelo sistema no momento seja o módulo GPRS/GSM. Quando a função 'readGprs()' detecta uma resposta do módulo em comando AT com o trecho '+CIPGSMLOC', significa que houve um retorno com a localização atual do dis-

positivo, executando então a função 'saveLastVehicleStatusOnCloud()', que se utiliza das funções de chamadas HTTP para enviar a atual situação do veículo (localização atual, última situação da tranca do veículo e evento detectado). Esta função é chamada quando se detecta algum movimento pelo sensor de presença PIR, além dos eventos de veículo trancado e destrancado, quando um dispositivo autorizado se conecta ou se desconecta do ponto de acesso WiFi, especificando antes do envio o evento no momento na variável 'lastMomentAction' que é usada como parâmetro no corpo da requisição. Atualizações automáticas de localização são enviadas a cada 30 minutos pela função 'sendLastLocationIfNeeded()' que é executada na função de repetição 'loop()'. A todo momento a função 'sendLastLocationIfNeeded()' verifica se já se passaram 30 minutos após o último envio de localização, e atribui à variável 'lastMomentAction' um número inteiro que é identificado por não haver evento no momento.

Conclui-se a etapa de implementação do serviço em nuvem. Em seguida, nas seções 4.2.16 a 4.2.20 são descritas as etapas de desenvolvimento do aplicativo em iOS, que interage com o sistema de notificações e permite a monitoração pelo usuário.

4.2.16 Criação da base da aplicação e preparação para comunicação com a nuvem

A versão utilizada da IDE Xcode para desenvolvimento da aplicação do sistema, é a 7.3 (7D175). Ao criar um projeto, escolheu-se uma base pré-definida pela IDE, chamada de 'Single View Application'. Como mostra a figura 4.27, a linguagem escolhida foi Objective-C, e as opções 'Core Data', 'Include Unit Tests' e 'Include UI Tests' foram desmarcadas.

A interface do aplicativo foi criada inteiramente no tipo 'Storyboard', um documento que permite manipulação dos elementos de modo visual e organização das telas baseado em fluxo.

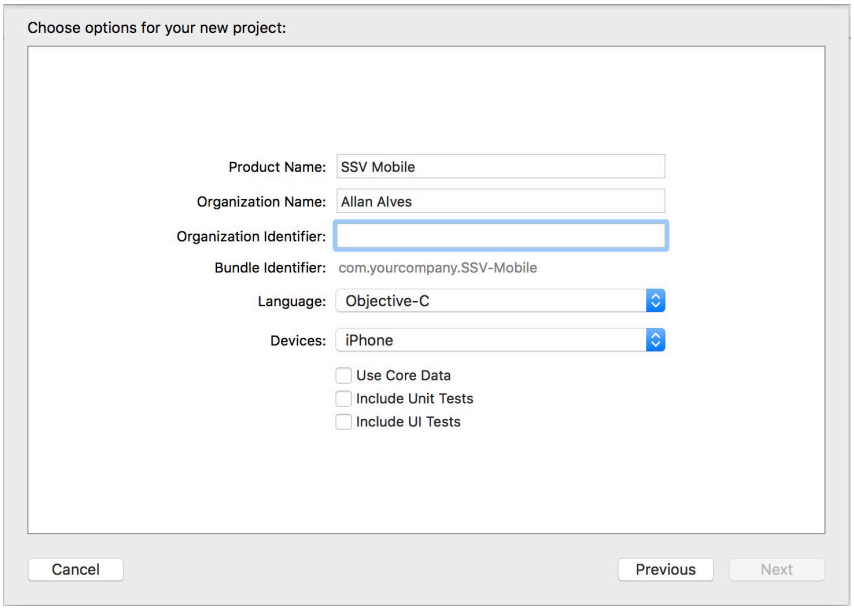


Figura 4.27 – Tela de criação de projeto da IDE Xcode. (Autor)

Utilizou-se o framework 'MapKit', para a implementação de mapas em telas que mostram localizações registradas do dispositivo de segurança. Este framework foi adicionado por meio da seção 'Linked Frameworks and Libraries' localizada nas configurações do projeto. O framework responsável por integrar o serviço do Parse com o aplicativo, pode ser encontrado no endereço: `https://github.com/ParsePlatform/Parse-SDK-iOS-OSX` (Acesso em 26/05/2016), e pode ser instalado de modo manual (copiando os arquivos para a raíz de diretórios do projeto), ou utilizando uma ferramenta de gerenciamento de dependências. O meio de instalação utilizado foi a ferramenta 'Cocoapods', que pode ser encontrada no endereço: `https://cocoapods.org/` (Acesso em 26/05/2016). Para instalar a ferramenta gerenciadora de dependências e em seguida utilizá-la para incluir o framework 'Parse' no projeto, segue-se os seguintes passos:

1. Abrir o terminal em uma máquina com ambiente Mac OS X ou baseado em Linux/Unix.

2. Utilizar o seguinte comando para a instalação da ferramenta:

```
sudo gem install cocoapods
```

3. Inserir a senha do usuário administrador e prosseguir. Após algum tempo, espera-se que seja apresentada a mensagem de conclusão:

```
13 gems installed.
```

4. Acessar o diretório raíz do projeto do aplicativo criado anteriormente (no caso '/Users/allanalves/Documents/SSV Mobile').

5. Utilizar o comando para a criação do arquivo 'Podfile':

```
pod init
```

6. Abrir para editar o arquivo 'Podfile' e incluir a seguinte linha abaixo de '# Pods for SSV Mobile':

```
# Pods for SSV Mobile
pod 'Parse'
```

7. Executar o seguinte comando no terminal:

```
pod install
```

8. Aguardar a seguinte mensagem de conclusão:

```
Re-creating CocoaPods due to major version update.
Analyzing dependencies
Downloading dependencies
Installing Bolts (1.7.0)
Installing Parse (1.13.0)
Generating Pods project
Integrating client project
Sending stats
Pod installation complete! There are 1 dependency from the Podfile and 2 total
pods installed.
```

Outro framework utilizado é o chamado 'iOS System Services', criado por 'Shmoopi LLC', que pode ser encontrado no endereço: `https://github.com/Shmoopi/iOS-System-Services` (Acesso em 26/05/2015). Utiliza-se este framework para identificação de endereço IP do roteador que o smartphone encontra-se conectado no momento em que é executado. Para instalar segue-se os mesmos passos anteriormente descritos (a partir do passo 6, se todos já foram realizados), adicionando a linha 'SystemServices' no arquivo 'Podfile'.

4.2.17 Recebimento de informações da nuvem pelo aplicativo

Para salvar e buscar informações armazenadas no servidor, utiliza-se o framework instalado 'Parse'. É incluido em arquivos de código (como 'AppDelegate.h' e 'StatusViewController.h' por exemplo) que o utilizam para fazer chamadas de manipulação de objetos ou para definir suas configurações iniciais. No 'Application Delegate' ('AppDelegate.h'), importa-se o arquivo 'header' (.h) do framework Parse:

```
#import <Parse/Parse.h>
```

No arquivo 'AppDelegate.m', configura-se a utilização do framework indicando a chave de identificação da aplicação 'API Key' e a chave 'Client Key' (definidas no arquivo 'PrefixHeader.pch'):

```
//Initialize Parse
[Parse setApplicationId:PARSE_APP_ID clientKey:PARSE_CLIENT_KEY];
```

Após definidas as chaves para integração com o Parse, qualquer 'View Controller' pode utilizar as classes do framework para efetuar chamadas para buscar e salvar objetos. Os métodos que implementam as chamadas que buscam objetos do serviço são criados na classe 'General.h'. O algoritmo 4.9 mostra o método para buscar todos os objetos da classe do serviço 'VehicleStatus', para serem listados na tela de histórico de situação do veículo ('StatusHistoryViewController').

Algoritmo 4.9 – Método que busca todos os objetos de situação do veículo. (Autor)

```
+ (void)requestAllVehicleStatusWithCompletion:(CompletionBlock)completion {
   PFQuery *query = [PFQuery queryWithClassName:kParseTableVehicleStatus];
   [query orderByDescending:@"createdAt"];
   [query findObjectsInBackgroundWithBlock:^(NSArray * _Nullable objects, NSError * _Nullable
       error) {
      if (error) {
         completion(NO);
      } else {
         SharedGeneral.allVehicleStatus = objects;
         completion(YES);
      }
   }];
}
```

No algoritmo 4.9, a classe 'PFQuery' cria uma consulta ao banco ao pelo método 'findObjectsInBackgroundWithBlock', e retorna todos os objetos de situação do veículo

ordenados descendentemente por data de criação. Os objetos recuperados são da classe 'PFObject', que vem com suas propriedades de acordo com seus dados salvos na nuvem.

4.2.18 Criação da interface de apresentação ao usuário

Cada tela criada, referencia dois arquivos (dos tipos com final .h e .m) que representam um chamado 'View Controller', onde se encontra o código do funcionamento de cada tela. Estes arquivos, como por exemplo 'StatusViewController.h' e 'StatusViewController.m', podem ser encontrados no apêndice D.

A estrutura da interface criada se baseia nas seguintes telas e seus respectivos 'View Controllers':

- Situação do Veículo - 'StatusViewController' (figura 4.28): mostra a última situação registrada no servidor. Nesta tela pode-se optar por atualizar para buscar novamente e ver o histórico de situações. Pode-se visualizar no mapa a localização da última situação registrada.

Figura 4.28 – Tela de situação do veículo: situação mais recente registrada no servidor. (Autor)

- Histórico - 'StatusHistoryViewController' (figura 4.29): mostra a lista do histórico de situações registradas no servidor. Pode-se visualizar a localização no mapa de uma situação selecionada na lista.

Figura 4.29 – Tela de histórico: lista dos registros de situação do veículo. (Autor)

- Dispositivos Autorizados - 'DevicesTableViewController' (figura 4.30): mostra a lista de dispositivos adicionados para obter acesso ao veículo por destravamento das trancas e desativação da alerta, emitida por movimento detectado.

Figura 4.30 – Tela de dispositivos autorizados: lista com os dispositivos autorizados para utilizar funções do sistema. (Autor)

- Detalhes do Dispositivo - 'DeviceDetailsTableViewController' (figura 4.31): mostra o nome, modelo e endereço MAC de um dispositivo selecionado. Pode-se excluir o dispositivo da lista.

Figura 4.31 – Tela de detalhes de dispositivo: mostra nome, modelo e endereço MAC de um dispositivo selecionado da lista. (Autor)

- Adicionar Dispositivo - 'AddDeviceTableViewController' (figura 4.32): tela com formulário com campos de nome, modelo e endereço MAC para adicionar um dispositivo na lista. O botão ao lado do campo 'Endereço MAC', orienta ao usuário copiar o endereço MAC nos ajustes do sistema, para assim.

Figura 4.32 – Tela de adicionar dispositivo: adiciona um dispositivo autorizado na lista usando as informações dos campos de texto disponíveis. (Autor)

- Controle - 'ControlTableViewController' (figura 4.33): pode-se optar por ações como acionar o controle de alarme, solicitar para o dispositivo sincronizar lista de dispositivos autorizados e atualizar localização. As ações podem ser acionadas quando o smartphone utilizado estiver conectado no ponto WiFi do dispositivo de segurança.

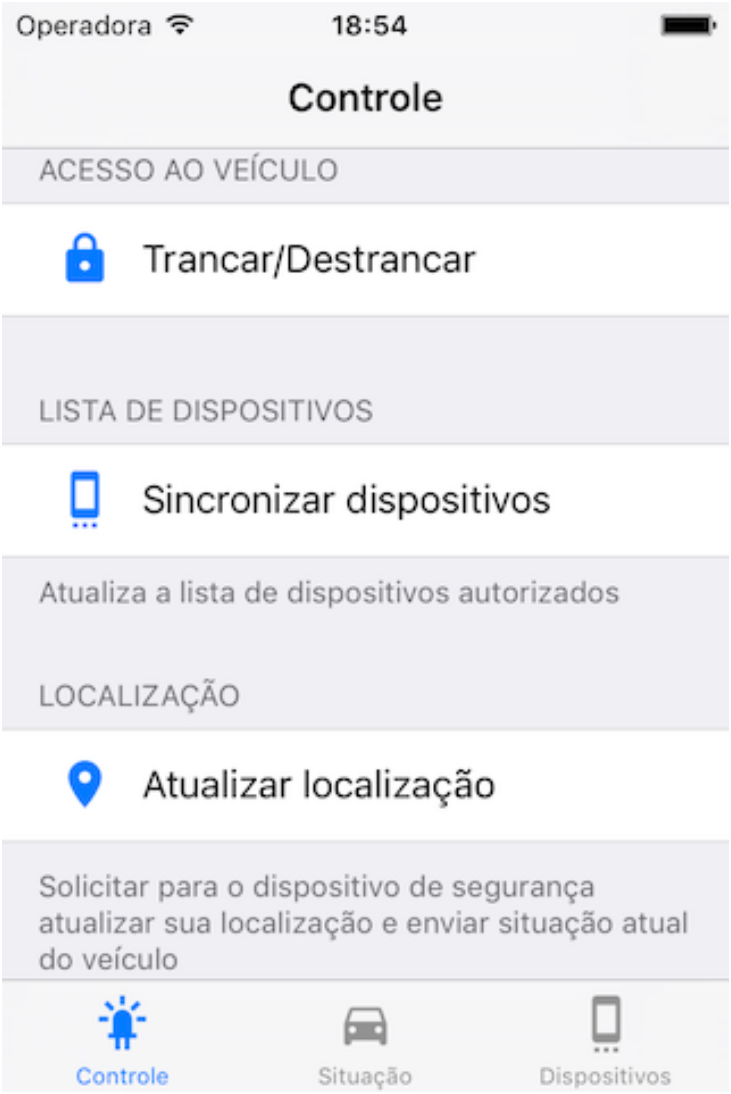


Figura 4.33 – Tela de controle: aciona a ação selecionada dentre as listadas. (Autor)

As telas mostradas são desenvolvidas no arquivo 'Main.storyboard' (apêndice D), onde manipula-se o fluxo do aplicativo, suas características de aparência e ligação aos arquivos 'View Controller', pela área 'Canvas' do Xcode. A base das telas do aplicativo, é do tipo 'Tab Bar Controller', que contém os 'View Controllers' de modo que se tenha uma organização semelhante a uma estrutura em árvore, tendo três itens na parte inferior da tela: 'Situação', 'Dispositivos' e 'Controle', que levam ás suas respectivas telas. Cada 'View Controller' tem em seu nível superior, um 'Navigation Controller', que permite adicionar botões de controle e título na parte superior da tela, assim como criar uma um fluxo navegação linear. A figura 4.34 representa o fluxo criado para a seção de situação do veículo. A primeira tela representa o 'Tab Bar Controller', que contém um 'Navigation Controller', que tem como fluxo linear os 'View Controllers' chamados de

'StatusViewController' e 'StatusHistoryViewController'. A tela de histórico é chamada quando o usuário seleciona o botão na parte superior da tela de situação do veículo.

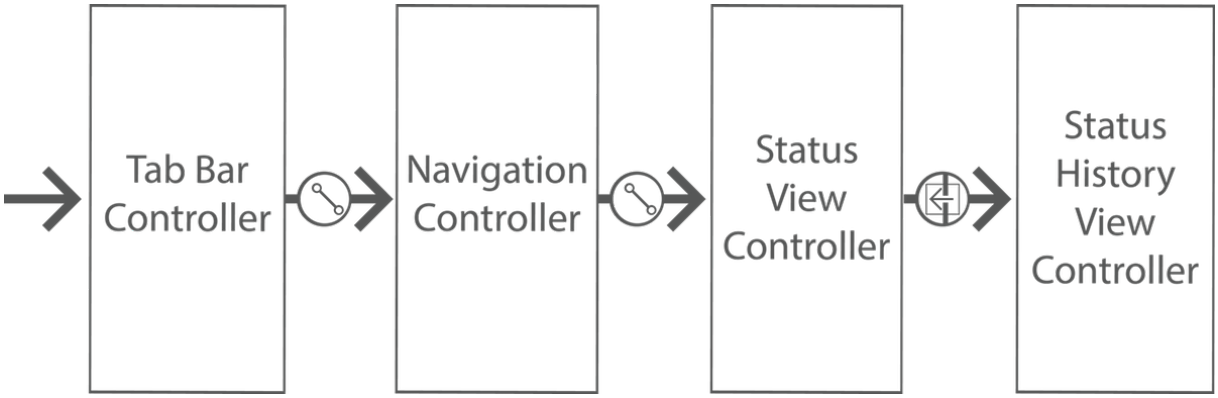


Figura 4.34 – Fluxo das telas relacionadas à situação do veículo. (Autor)

4.2.19 Preparação para recebimento de notificações pelo aplicativo

Para ativar a capacidade do aplicativo de receber notificações push remotas, opta-se nas configurações do projeto, em 'Capabilities', a opção 'Push Notifications', como mostra a figura 4.35.

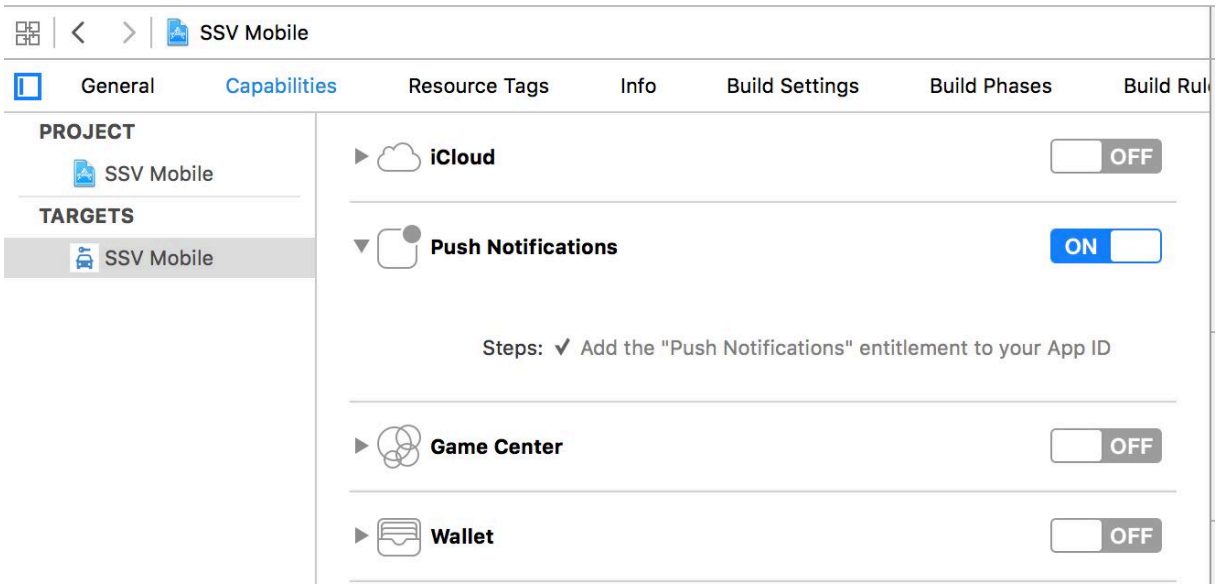


Figura 4.35 – Xcode - tela de escolha de capacidades do aplicativo. (Autor)

Em uma conta de desenvolvimento da Apple, no sítio `https://developer.apple.com/account/` (Acesso em 26/05/16), seleciona-se a opção 'Certificates, Identifiers &

Profiles'. Em 'App IDs', na seção 'Identifiers', pode-se adicionar e editar configurações relacionadas aos aplicativos. Para ativar as notificações, é preciso adicionar um ID de aplicativo, selecionando o botão '+'. Marca-se a opção 'Push Notifications', e então em 'Apple Push Notification service SSL Certificates', cria-se um certificado de desenvolvimento, selecionando o botão 'Create Certificate...'. Para criá-lo é preciso gerar um arquivo do tipo CSR (.certSigningRequest), Certificate Signing Request. Para gerá-lo deve-se abrir a aplicação 'Keychain Access' em um ambiente Mac OS X, acessar o item de menu: 'Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority...'. Nos campos, informa-se um endereço de e-mail, nome e marca-se a opção 'Saved to disk'. Ao continuar, seleciona-se o diretório para salvar o arquivo. Novamente na tela de criação de certificado para notificações push no sítio da Apple citado anteriormente, seleciona-se o arquivo no diretório em que foi gerado. Seleciona-se 'Continue' e 'Download', para então abrir o arquivo e registrá-lo na máquina.

É necessário incluir o certificado nas configurações do aplicativo adicionado no sítio do Parse (`http://www.parse.com/`). O arquivo necessário é do tipo p12, que pode ser gerado pela aplicação 'Keychain Access': na seção 'Category', 'Certificates', procurando por 'SSV', deve-se encontrar o item 'Apple Development IOS Push Services: AA.SSV-Mobile-20973560' (figura 4.36), clicar com o botão direito e selecionar a opção 'Export "Apple Development IOS Push Services: AA.SSV-Mobile-20973560"...'.

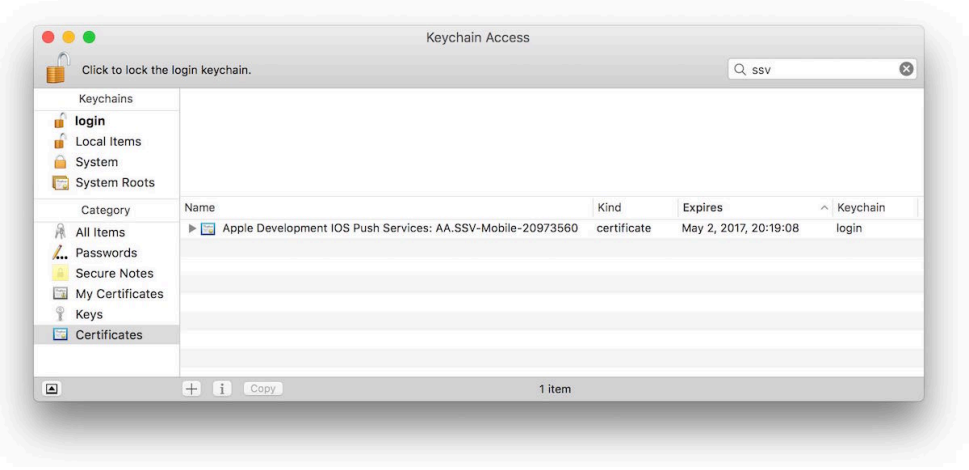


Figura 4.36 – Keychain Access - certificado de notificações push do aplicativo. (Autor)

Após salvar o arquivo do tipo p12 exportado, deve-se carregá-lo no sítio do Parse. Em 'Settings', 'Push', na seção 'Apple Push Certificates', seleciona-se a opção 'Select

your certificate', e escolhe-se o arquivo p12 exportado anteriormente. Com estes procedimentos já é possível certificar-se de que o aplicativo tem a capacidade de receber notificações push.

4.2.20 Implementações adicionais para envio de notificações aos aplicativos

No arquivo 'app.js', que executa a manipulação por trás das requisições por parte do servidor, são necessárias algumas funções para envio de notificações para o aplicativo instalado nos dispositivos autorizados. A função base para envio de qualquer tipo de notificação é mostrada no algoritmo 4.10.

Algoritmo 4.10 – Função que envia notificações push para os dispositivos autorizados. (Autor)

```
function sendPush (msg, type) {
  var sound;
  if (type == 0) sound = ""; //No Sound
  if (type == 1) sound = "default"; //Normal Sound
  if (type == 2) sound = "alarm.wav"; //Alarm Sound
  Parse.Push.send({
    channels: [ "global" ],
    data: {
      sound: sound,
      alert: msg
    }
  }, {
    success: function() {
      // Push was successful
      console.log("Push notification sent to all devices.");
    },
    error: function(error) {
      // Handle error
      console.log("Error trying to send push notification to all devices.")
    }
  });
}
```

Ao chamar a função, passa-se a mensagem e o tipo de som que será emitido no dispositivo que receber a notificação. O arquivo de som nomeado de 'alarm.wav', tem um volume mais alto e denota alerta. Já o 'default' é o padrão do dispositivo. A notificação enviada por essa função é direcionada para o canal nomeado de 'global', que envia a todos os dispositivos que foram automaticamente registrados na classe 'Installation' ao instalar e executar o aplicativo.

A função que retorna os endereços MAC dos dispositivos autorizados, utiliza esta função de envio, para informar que a lista foi sincronizada e o dispositivo físico foi inicializado com sucesso.

4.2.21 Implementação do envio de comandos para acionamento de certas funções do sistema

Ao selecionar um botão na tela de controle do sistema (figura 4.33), efetua-se uma comunicação do tipo TCP/IP com o IP do roteador, que no caso seria o módulo WiFi, como mostrado no algoritmo 4.11.

Algoritmo 4.11 – Método que cria conexão com o módulo WiFi e envia um comando específico. (Autor)

```
- (void) sendRequestWithCommand:(NSString*)command {
   [self updateWifiIP];

   NSString *url = [NSString stringWithFormat:@"http://%@:80", wifiRouterIP];

   NSString *post = [NSString stringWithFormat:@"comm=%@", command];
   NSData *postData = [post dataUsingEncoding:NSUTF8StringEncoding];

   NSMutableURLRequest *request = [[NSMutableURLRequest alloc] init];
   [request setURL:[NSURL URLWithString:url]];

   [request setHTTPMethod:@"POST"];

   [request setHTTPBody:postData];

   NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:request
       delegate:self];

   if(connection) {
      NSLog(@"Connected");
   } else {
      NSLog(@"Not Connected");
   }
}
```

O código completo da tela de comandos pode ser encontrado no apêndice D, no arquivo 'ControlTableViewController.m'. A requisição enviada é do tipo 'post', com o parâmetro 'comm' no corpo da requisição. Este parâmetro é enviado com o valor numérico do comando. Os comandos definidos foram:

- Código 991: controle de acesso ao veículo. Aciona trancamento ou destrancamento na placa de controle de alarme.
- Código 992: solicitação de sincronização de lista de dispositivos autorizados. Envia a lista de dispositivos para o servidor.
- Código 993: solicitação de atualização de situação do veículo e sua localização. Registra as informações no servidor, no momento em que é acionado.

O método 'updateWifiIP' chamado no início do algoritmo mostrado anteriormente, detecta o endereço IP do roteador em que o smartphone está conectado.

Todas as etapas descritas neste capítulo foram necessárias para o perfeito funcionamento da solução. O código completo da solução é apresentado no apêndice B. A solução será apresentada com testes e resultados no próximo capítulo.

5 Aplicação do Modelo Proposto

5.1 Apresentação da Área de Aplicação do Modelo

O sistema apresentado neste trabalho pode ser parcialmente integrado a uma solução de sistema de segurança para veículos. Como alternativa, se devidamente tratada, pode ser considerada em uma solução com aplicativo móvel para segurança veicular em conjunto com acionamentos de variadas partes automatizadas do veículo, como limpador de vidro, controle de ignição, teto solar, ou vidro elétrico.

5.2 Descrição da Aplicação do Modelo

5.2.1 Construção do equipamento e implementação de hardware

Em várias etapas da implementação física do projeto, foi necessário o uso da placa conversora USB para serial. A praticidade de optar pela tensão a ser trabalhada e a variedade de pinos, permitiu que fosse facilmente utilizada nas placas que necessitam de comunicação serial com um computador, viabilizar sua implementação.

Após fazer a montagem do módulo WiFi para efetuar comunicação serial, testou-se as versões de firmware 0.9.2.2 e 0.9.5(b1). Notou-se que a versão mais antiga, 0.9.2.2, apresentou em alguns testes uma falha, não detectando a desconexão do

único dispositivo conectado ao módulo WiFi. Esta falha causa o mal funcionamento de um evento importante no sistema, que seria a não ativação do controle de alarme do veículo para trancá-lo quando não houver nenhum dispositivo autorizado conectado ao módulo. Por este motivo, a versão 0.9.5(b1) foi escolhida para uso, por não apresentar a falha nos testes realizados.

No processo de integração do módulo WiFi ao Arduino, existe a necessidade de utilizar a tensão de 3,3V para alimentar o módulo WiFi. Para supri-la utilizou-se o regulador LM1117T-3.3.

A conexão serial para implementação do Arduino Pro Mini foi notada como estável e sem nenhum problema. A taxa escolhida para comunicação é de 115200 bps por apresentar um bom tempo de resposta. A taxa de 57600 escolhida para a placa Arduino se comunicar com o módulo WiFi, foi escolhida devido a falhas observadas (caracteres não esperados) em comunicação com taxas maiores em testes realizados.

Em testes feitos de comunicação entre a placa Arduino e o módulo WiFi, notou-se uma certa instabilidade no funcionamento do módulo WiFi. O módulo tem como reação à incorreta forma de alimentação, a sua reinicizalização automática. O uso de um capacitor de 470 Microfarads próximo aos conectores VCC e GND do módulo, agregou mais estabilidade ao módulo, apresentando o comportamento observado anteriormente com menos frequência.

Pela necessidade de se utilizar 5V no módulo GPRS/GSM, utilizou-se o regulador de tensão LM2596. Sua função no circuito é regular sua tensão de entrada de aproximadamente 12V a 14V provinda de um veículo, para 5V, tensão utilizada por alguns outros componentes citados ao longo do trabalho. O esquemático do apêndice A, mostra sua integração ao circuito.

Assim como a situação da comunicação entre a placa Arduino e o módulo WiFi mostrou-se problemática devido à alta taxa de transferência de dados (115200 bps), optou-se por utilizar a taxa de 57600 bps, tendo bons resultados após a mudança. Optou-se por utilizar um capacitor de 470 Microfarads próximo aos conectores VCC e GND do módulo GPRS/GSM, assim como utilizado para o módulo WiFi pelos mesmos problemas de estabilidade.

Verificou-se que o uso de uma quantidade alta de memória dinâmica da placa Arduino (aproximadamente 70%), provoca instabilidades no funcionamento, causando

falhas. Resolveu-se diminuindo o uso de variáveis do tipo 'String' e mensagens de debug. Limitou-se também a lista de dispositivos autorizados, permitindo somente dois, por causar necessidade de uso de alta quantidade de espaço de memória dinâmica da placa. Após aplicar as práticas para evitar instabilidades, o código utiliza em torno de 58% da memória dinâmica da placa.

Ao inicializar o módulo GPRS/GSM, são utilizados alguns comandos para configurá-lo para efetuar conexões pela Internet. Ao utilizar o comando 'AT+SAPBR=1,1', que tem como função abrir a conexão com a portadora, notou-se que é necessário fazer algumas tentativas, por precisar esperar um certo tempo para ocorrer a conexão. Preparado para isso, o sistema envia o comando novamente, até que normalmente, na terceira tentativa a conexão é realizada, segundo a maior parte dos testes realizados. Em uma tentativa falha, o módulo envia uma resposta informando um erro, como '+CME ERROR' ou 'ERROR'.

O mesmo comportamento de funcionamento instável das placas WiFi e GPRS/GSM devido à incorreta alimentação foi observado ao longo da implementação. Tratado como o principal motivo de instabilidade, o problema foi resolvido utilizando uma alimentação que provê um nível mais alto de corrente. A utilização de uma fonte de 3A para realização de testes resolveu por completo o problema notado. O regulador de tensão utilizado para converter a entrada de tensão vinda do veículo para 5V, limita a corrente passante no circuito para 2,5A a 3A, porém foi considerada suficiente após a realização de testes.

Notou-se que o regulador de tensão de entrada do dispositivo (LM2596) mantém o nível ajustado (5V) independentemente da variação da tensão do conector instalado no veículo para alimentar a placa. Em tensões mais altas que passam de 14V quando o veículo está ligado por ignição, verificou-se estabilidade na tensão de saída do regulador. A saída do regulador de tensão que recebe 5V e provê 3,3V também não foi influenciado pelo funcionamento do veículo.

O uso do 'buzzer' nos testes pós-implementação foi essencial para apoio nos testes em detecções e eventos lógicos do sistema.

Foi verificado que por se utilizar da tecnologia GSM para identificar a localização do dispositivo de segurança, não é possível obter uma localização com boa acurácia, devido a técnica de triangulação de torres utilizada na rede.

Os testes da funcionalidade de acionamento das travas por conexão ou desconexão ao ponto de acesso WiFi, foram realizados em um estacionamento de automóveis, utilizando dois aparelhos smartphones de marcas e sistemas operacionais distintos. O teste de aproximação foi realizado em um caminho livre de obstáculos físicos para não influenciar nos resultados. Verificou-se que a distância necessária para o reconhecimento automático do sinal do ponto de acesso se aproximava de 5 metros, variando de acordo com o aparelho. Nos testes de distanciamento, o acionamento para trancamento do veículo aconteceu em distâncias diferentes.

com um mínimo de 32 metros, podendo chegar a mais de 100 metros por continuar a existir uma conexão ativa entre o smartphone e o ponto de acesso, mesmo com sinal ainda bastante baixo.

Dependendo da marca ou do sistema operacional do smartphone, é possível não haver conexão automática por não possuir conexão à Internet disponível no ponto de acesso, sendo detectado, porém ignorado pelo sistema operacional.

Para que o sensor de presença (PIR) não detecte movimento em um ângulo de visão obtuso, ou seja, com mais de 90 graus, e tenha o funcionamento próximo do esperado, utilizou-se uma pequena peça de borracha para envolver o sensor para direcionar a identificação de movimentos de modo que acione dentro de um ângulo menor. Nos testes da funcionalidade de detecção de instrusão no veículo foram realizados três testes para avaliar a estabilidade e o funcionamento do sensor PIR para detecção de movimento interno. Todos foram realizados após o veículo ser trancado por meio de desconexão de qualquer smartphone autorizado do ponto de acesso WiFi. O comportamento do sensor PIR foi observado com os seguintes posicionamentos do componente no veículo:

- Embaixo do volante e voltado para o piso, de modo que detecte movimento des pernas de um indivíduo sentado no assento do motorista (figura 5.1).

Figura 5.1 – Sensor PIR posicionado abaixo do volante voltado para o piso do veículo. (Autor)

- Ao lado do câmbio, de modo que aponte para a direção das pernas de um indivíduo sentado no assento do motorista (figura 5.2).

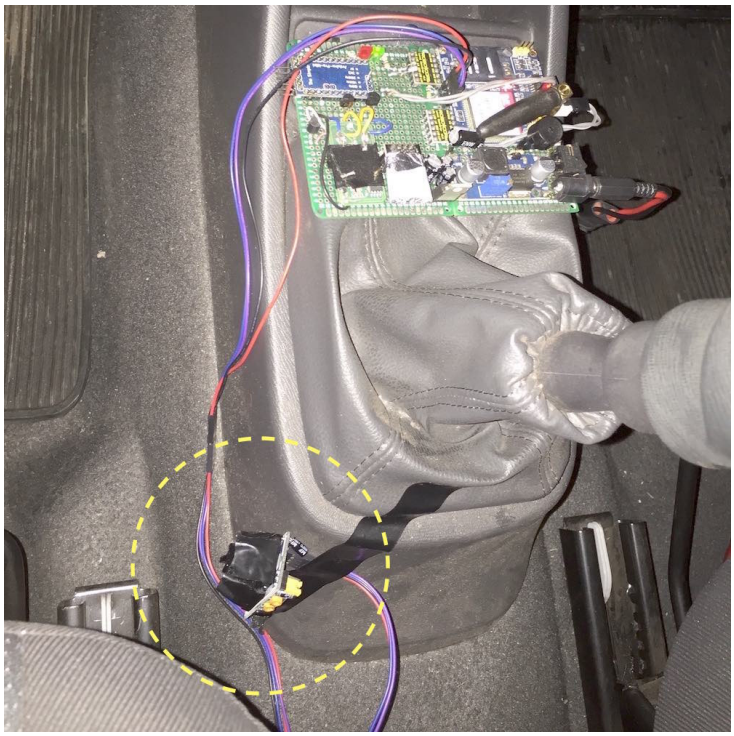


Figura 5.2 – Sensor PIR posicionado ao lado do câmbio do veículo. (Autor)

- Próximos aos pedais, de modo que aponte para a direção dos pés de um indivíduo sentado no assento do motorista (figura 5.3).

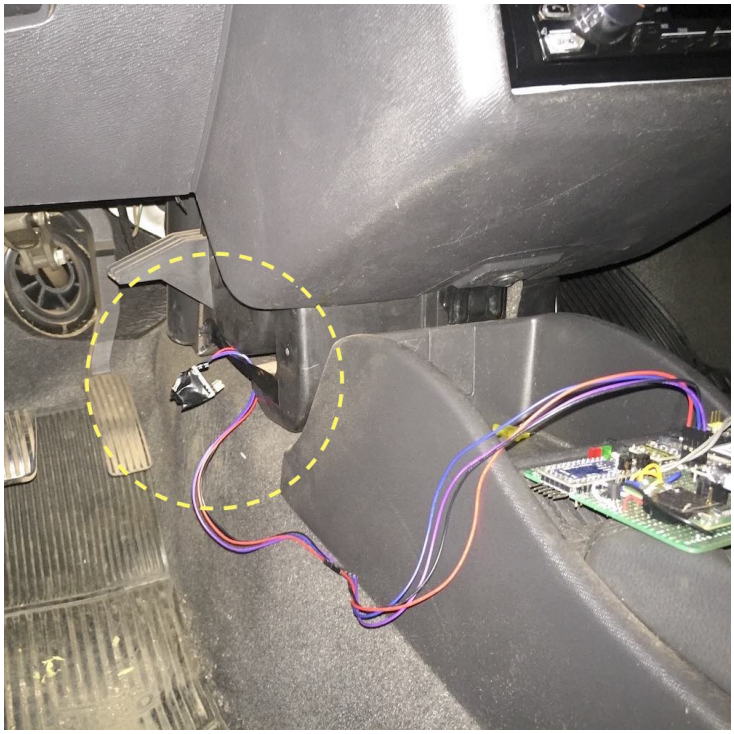


Figura 5.3 – Sensor PIR posicionado próximo aos pedais do veículo. (Autor)

O período de teste após o trancamento do veículo foi de cinco minutos. Após os testes de observação foram realizadas simulações de intrusão, entrando no veículo e se posicionando no assento do motorista, após destrancar o veículo com um controle de alarme externo, ou seja, sem acionamento no sistema, tendo o mesmo efeito de uma intrusão no ponto de vista do funcionamento lógico do sistema. Tendo como base o funcionamento ideal, que seria a detecção de movimento apenas quando algum indivíduo entra no veículo e se posiciona no assento do motorista, foram extraídos os seguintes resultados com os respectivos posicionamentos do sensor:

- Embaixo do volante: uma detecção inesperada nos primeiros segundos e cinco em momentos aleatórios. Funcionamento esperado na simulação de intrusão.
- Ao lado do câmbio: uma detecção inesperada nos primeiros segundos. Funcionamento esperado na simulação de intrusão.
- Próximo aos pedais: Nenhuma detecção inesperada. Funcionamento esperado na simulação de intrusão.

Devido aos repetitivos envios desnecessários por detecção de movimento, optou-se por utilizar a variável booleana 'alreadyDetectedMotionAfterLock' para evitar o envio de mais de um alarme de suspeita de invasão. A variável tem o valor padrão como falso, por ainda não ter enviado o alarme, e quando aciona o comando de trancamento do veículo, muda-se o seu valor para verdadeiro, tornando a ser falso quando um movimento é detectado.

5.2.2 Implementação do serviço de comunicação e armazenamento de informações em nuvem

Testes durante a implementação foram realizados enviando requisições aos endereços preparados e analisando suas respostas. A ferramenta do serviço 'Parse' chamada 'Cloud Code' possui registro constante de atividade e inclui certos detalhes de erros, mensagens de aviso ou informações a nível de código, ao utilizar a função 'console.log()', por exemplo, como pode se observar em alguns trechos do código completo (apêndice C).

No trecho em que se prepara a chamada do tipo 'post' pelo endereço '/macDevices', que reune os endereços MAC dos dispositivos autorizados derivados do banco de dados, utiliza-se uma técnica para resumir a resposta para o recebimento do dispositivo de segurança. Pela necessidade de evitar alta utilização da memória dinâmica da placa Arduino, como citado anteriormente, os endereços são tratados de modo que sejam enviados sem a pontuação padrão (caractere ':'). Para isso, limita-se também a quantidade de dispositivos para dois, sendo o primeiro o dispositivo a ser utilizado normalmente e o segundo como um dispositivo reserva, caso haja impossibilidade de se utilizar o principal.

Nos testes de envio e recebimento, notou-se que o módulo GPRS/GSM leva alguns segundos para enviar uma requisição, que adicionando com o tempo de resposta do serviço em nuvem e o recebimento das notificações nos smartphones, é considerado irrelevante, levando de 4 a 10 segundos. O tempo é contado desde o início de um comando que solicita o envio de uma requisição até o recebimento de uma notificação nos smartphones.

O serviço MBaaS utilizado no projeto, de certa forma reduziu o esforço e tempo consideravelmente, já que não havia intuito de dar ênfase em detalhes deste meca-

nismo do sistema de segurança. Por sua utilização, não foi preciso o uso e a implementação de tecnologias para envio de notificações, criação, gerenciamento e armazenamento de dados, rotas para chamadas e envio de respostas, e hospedagem para implementação do servidor. O serviço resume todas estas necessidades em simples interações usando uma aplicação para gerenciamento geral.

5.2.3 Desenvolvimento da aplicação para smartphone e interação com o sistema de notificações

A implementação dos controles principais (acionamento de travas, pedido de sincronização de lista de dispositivos e pedido de envio de situação e localização) por botões no aplicativo, obteve resultados positivos ao efetuar testes com um dispositivo conectado ao módulo. Ao efetuar testes com mais de um dispositivo, pode-se sobrecarregar o módulo com comandos, se enviados no mesmo momento, tendo em conta que a placa Arduino envia a cada dois segundos um comando para verificar os dispositivos conectados ao módulo WiFi, possuindo isso como uma atividade constante. Como a necessidade ideal e esperada é conectar somente um dispositivo para controle de um veículo pessoal, este problema de possível sobrecarga torna-se irrelevante para este projeto.

Nos testes de envio de comandos para o módulo WiFi, notou-se que quando a conexão celular (2G ou 3G) está ativa, o método que identifica o endereço IP do roteador (’[[SystemServices sharedServices] wiFiRouterAddress]’) retorna um endereço IP não pertencente à rede gerada pelo módulo WiFi. Para contornar este problema, utilizou-se o método para identificar o endereço IP do smartphone na rede (’[[SystemServices sharedServices] wiFiIPAddress]’), modificando o último componente do endereço para ’1’, passando a ter um endereço do tipo ’192.168.4.1’ caso o endereço do smartphone na rede do ponto de acesso for ’192.168.4.2’ por exemplo.

5.3 Custos do Modelo Proposto

A tabela 5.1 mostra a quantidade e o valor de cada componente utilizado no projeto.

Tabela 5.1 – Custos dos componentes utilizados no projeto

Componente	Quantidade	Custo (unidade)
ESP8266 ESP-01	1	R$ 27,95
SIM900A Mini	1	R$ 195,00
Placa Perfurada 6 cm x 8 cm	2	R$ 4,95
Regulador de Tensão LM2596	1	R$ 14,90
Conversor de nível lógico	1	R$ 8,50
Regulador de tensão LM1117T-3.3	1	R$ 5,39
Capacitor 10 microfarads	2	R$ 0,99
Capacitor 470 microfarads	2	R$ 1,11
Placa FTDI FT232RL	1	R$ 34,90
Transistor PNP BC327-25	3	R$ 0,14
Sensor PIR	1	R$ 14,46
Arduino Pro Mini 328 - 5V/16MHz	1	R$ 27,60
Led Difuso Retangular Verde/Vermelho	2	R$ 0,22
Chave Gangorra 2 terminais	1	R$ 0,79
Buzzer 5V	1	R$ 1,37
Resistor 10K Ohms	2	R$ 0,04
Resistor 300 Ohms	2	R$ 0,04
Jack J4 DC-005	1	R$ 0,43
Conector Plug P4	1	R$ 0,69
Kit de conectores empilháveis	1	R$ 4,68
Cabo 4 Pinos Fêmea-Fêmea	1	R$ 5,90
Cabo 4 Pinos Macho-Fêmea	1	R$ 5,90
TOTAL		R$ 363,58

5.4 Avaliação Global do Modelo

O desenvolvimento do dispositivo de segurança é tido como uma aplicação básica de funcionalidades atreladas a um aplicativo móvel para ser utilizado como um adicional quando se tem alarme e travas automáticas instaladas no veículo, tornando possível uma simples monitoração. O trabalho não tem como objetivo desenvolver um sistema que evite ou reduza as chances de que algum incidente ou intrusão possa acontecer, se limitando somente na monitoração e integração do sistema de alarme/-travas já existentes. A integração do controle de alarme apresentada neste trabalho não possibilita o seu uso para qualquer sistema de alarme, sendo apenas integrado a um veículo que já possui o sistema instalado e o controle já programado.

O sensor de presença passa a ideia de um modo de notificar o usuário do sistema, sendo somente um dos vários tipos disponíveis que poderiam ser integrados em seu lugar. A utilização da plataforma de prototipação Arduino, tornou possível integrar

todos os componentes do sistemas e unificar a lógica geral.

6 Conclusões

Após o completo desenvolvimento do sistema de segurança, pôde-se verificar a viabilidade da escolha de cada tipo de componente integrado, mesmo não tendo como foco a implementação individual de cada componente adotado (placa WiFi, placa GPRS/GSM, sensor PIR e controle de alarme). Verificou-se que em uma única placa pode-se adicionar variados tipos de funcionalidades, sendo independente de conexões ao sistema elétrico do veículo, com exceção do cabo de alimentação, podendo assim ser utilizado somente conectando este cabo.

Com a conclusão dos testes notou-se que a funcionalidade de detecção de intrusão, uma importante funcionalidade, merece ser cuidadosamente estudada para não haver ocorrência de alarmes falsos, como verificado nos resultados dos testes, que para se chegar a uma conclusão do posicionamento ideal do sensor, foram necessárias algumas tentativas com diferentes técnicas. Notou-se que o uso de um sensor de melhor acurácia seria uma melhora importante para o funcionamento ideal do sistema.

O acionamento de travas pelo aplicativo, pode tornar o sistema independente de um controle de alarme, tornado desnecessário levá-lo sempre junto às chaves do veículo, sendo então substituído por um smartphone autorizado registrado no sistema. Já o acionamento de outros comandos relativos à ações do sistema de segurança, são dependentes de um smartphone com o aplicativo desenvolvido neste trabalho instalado, limitando-se então ao sistema operacional iOS. Outros modelos de smartphone com estas funcionalidades limitadas, tem então a função de servir como um smartphone secundário, caso o principal for perdido ou sair da posse do usuário. É comum que a desconexão manual do smartphone com o ponto de acesso WiFi seja necessária quando o usuário se distancia do veículo, por precisar de um raio em que o usuário não consegue mais enxergar que o acionamento de trancamento foi realizado. A desconexão manual é de simples acesso nos smartphones, no sistema operacional móvel iOS, se acessa com o telefone travado, ou seja, com permissão de uso bloqueada protegida por senha. Por isso, a desconexão manual não seria um problema. Já o acionamento das travas ao se aproximar do veículo teve o funcionamento como esperado, ou seja, acionamento a uma distância curta.

O envio de localizações frequentes (a cada 30 minutos) teve uma importante função para fins de monitoração. Esta funcionalidade teve os resultados esperados.

Soluções modernas que tornam a implementação de um servidor para suprir necessidades de aplicativos e dispositivos com acesso à rede de Internet, podem ser utilizados para a realização de experimentos sem a necessidade de muito esforço, podendo evoluir após testes e análises de implementações.

Sobre o ponto de vista geral, o projeto teve sucesso em suas implementações e testes. As fucionalidades tiveram êxito no funcionamento.

6.1 Sugestões para Trabalhos Futuros

Tendo em conta que o dispositivo desenvolvido tem como característica principal ser móvel, e facilmente instalado em qualquer veículo, tornaria mais adaptável um sistema com o controle de alarme e travas programável para outros sistemas de alarme. Com a escolha de um componente emissor de sinal responsável por acionar o sistema de qualquer marca e modelo, ou os mais comuns dos disponíveis no mercado, integrar este componente, substituindo o atual.

O módulo utilizado permite o uso de múltiplas funções, como SMS, atendimento de ligações e recebimento de comandos. De modo semelhante a um sistema com URA, é possível controlar o sistema por uma simples ligação, digitando números para solicitar comandos e informar senha de acesso ao controle. Esta funcionalidade poderia se tornar uma alternativa para casos em que o usuário perde a posse de seu smartphone juntamente com o veículo ou não é possível seu uso devido à falta de bateria ou caso semelhante.

Uma implementação adicional que tornaria o sistema mais customizável pelo usuário, seria incluir preferências pelo aplicativo para a escolha de comportamentos automáticos do sistema, como por exemplo, permitir o acionamento automático das travas ou escolher o intervalo de tempo que o dispositivo registra sua localização no servidor frequentemente.

Explorar as opções disponíveis para a substituição do módulo GPRS/GSM, poderia ser um melhoramento positivo para o sistema, caso as características de um novo módulo incluírem mais rapidez no envio e recebimento de informações pela Internet.

O uso de Bluetooth, outro tipo de tecnologia sem fio, poderia ser estudada para ser integrada como opção de uso, além da tecnologia WiFi utilizada no projeto. A análise das vantagens e desvantagens de cada tecnologia poderia revelar a viabilidade da

implementação de uma delas ou ambas.

Estudo de diferentes técnicas e componentes a serem usados para detectar movimentos internos no veículo, como o sensor de ultrassom, que pode medir a distância entre um objeto próximo e o sensor, ajudando a evitar emissão falsa de alertas, utilizando-se da identificação de amostras incomuns dentre as coletadas.

Uso de um módulo que seja possível o uso da tecnologia GPS para garantir uma melhor acurácia na identificação da localização do dispositivo de segurança.

Desenvolvimento do aplicativo de monitoração para outros sistemas operacionais móveis (além do iOS adotado neste trabalho), tornando possível utilizar comandos para realizar qualquer ação do sistema de segurança, não limitando somente ao acesso por WiFi acionando as travas do veículo.

Referências Bibliográficas

1. HISAYASU, Alexandre - Estadão - A cada 2 minutos e meio, 1 carro é roubado nas 27 capitais do país - http://brasil.estadao.com.br/noticias/geral,a-cada-2-minutos-e-meio--1-carro-e-roubado-nas-27-capitais-do-pais,1773012 Acesso em 23/02/2016

2. BRAZ, Jorge - A Importância dos Smartphones no Marketing e nas Marcas - http://mediassociais.com/2012/09/06/a-importancia-dos-smartphones-no-marketing-e-nas-marcas/ Acesso em 10/03/2016

3. SUSEP, Aumenta índice de roubo de carros no país - http://www.susep.gov.br/setores-susep/noticias/noticias/aumenta-indice-de-roubo-carros-no-pais Acesso em 10/03/2016

4. SINESP, Estatísticas Criminais - https://www.sinesp.gov.br/estatisticas-publicas Acesso em 31/03/2016

5. SSP-RS, Secretaria de Segurança Pública do Rio Grande do Sul - http://www.ssp.rs.gov.br/ Acesso em 10/03/2016

6. MUL-T-LOCK - Trava de câmbio - http://www.multlock.com.br/produtos/trava-cambio-automotiva-veiculos-contra-arrombamento.php Acesso em 04/04/2016

7. GIOVANNETTI, Eduardo - A EVOLUÇÃO DA ELETRÔNICA EMBARCADA NA INDÚSTRIA AUTOMIBILÍSTICA BRASILEIRA, 2011 - http://maua.br/files/monografias/a-evolucao-da-eletronica-embarcada-na-industria-automobilistica-brasileira.pdf. Acesso em 09/04/2016

8. BIDU, Rastreador de carro: O que é e como funcionam os rastreadores veiculares - https://www.bidu.com.br/seguro-auto/guia-como-funciona-um-rastreador-veicular/ Acesso em 10/04/2016

9. CHIPTRONIC, Blog - Conheça a história da evolução das chaves de carro - http://chiptronic.com.br/blog/conheca-a-historia-da-evolucao-das-chaves-de-carro Acesso em 10/04/2016

10. MALUF, Flavio - A Importância dos aplicativos móveis dia-a-dia - http://noticias.r7.com/dino/tecnologia-e-ciencia/a-importancia-dos-aplicativos-moveis-no-dia-a-dia-comenta-flavio-maluf-28092015/ Acesso em 11/03/2016

11. APPLE, iOS Developer Library - Apple Push Notification Service - https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificat Chapters/ApplePushService.html/ Acesso em 03/05/2016

12. APPLE, Apple Press Info - http://www.apple.com/pr/library/ Acesso em 03/05/2016

13. APPLE, About Objective-C - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html Acesso em 05/05/2016

14. APPLE, AppKit Framework Reference - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/ObjC_classic Acesso em 05/05/2016

15. APPLE, Foundation Framework Reference - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/ObjC_classic/ Acesso em 05/05/2016

16. APPLE, Xcode Overview: At a Glance - https://developer.apple.com/library/tvos/documentation/ToolsLanguages/Conceptual/Xcode_Overview/index.html Acesso em 05/05/2016

17. ARDUINO, Introduction - What is Arduino? - https://www.arduino.cc/en/Guide/Introduction/ Acesso em 29/04/2016

18. ARDUINO, Arduino Pro Mini - https://www.arduino.cc/en/Main/arduinoBoardProMini Acesso em 04/05/2016

19. LOUZANO, F. A. - Desenvolvimento de novos métodos de detecção de movimento utilizando sensores infravermelho passivos, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, São Paulo, Brasil, 2010.

20. TOREYIN, B.U. & CETIN, E. E. Falling Person Detection Using Multi-Sensor Signal Processing, Department of Electrical and Electronics Engineering, Bilkent University, Bilkent, Ankara, Turkey, 2007.

21. IMOBILIS - ESP8266 Introdução E Primeiros Passos - http://www.decom.ufop.br/imobilis/esp8266-introducao-e-primeiros-passos/ Acesso em 07/05/2016

22. BADER, Daniel - iMore - What is a SIM card and what does it do? - http://www.imore.com/what-is-sim-card Acesso em 07/05/2016

23. DALAKOV, Georgi - History of Computers - The Modem of Dennis Hayes and Dale Heatherington - http://history-computer.com/ModernComputer/Basis/modem.html Acesso em 10/05/2016

24. PARANHOS, Felipe - Oficina da Net - Evolução do modem - https://www.oficinadanet.com.br/post/11873-evolucao-do-modem Acesso em 10/05/2016

25. USRobotics (Suporte) - Comandos de Modem - http://support.usr.com/support/3cxm756/3cxm756-portuguese-ug/atcoms.htm Acesso em 10/05/2016

26. PUJAR, Ravi - Difference between SIM900 and SIM900A GSM modems - http://www.raviyp.com/embedded/174-difference-between-sim900-and-sim900a-gsm-modems Acesso em 22/05/2016

27. CURVELLO, André - Embarcados - Apresentando o módulo ESP8266 - http://www.embarcados.com.br/modulo-esp8266/ Acesso em 10/05/2016

28. BENGTSSON, S. & ELIASSON, P. Development of Picwear - A Multi-platform Mobile Clothing and Fashion Service for Android, iOS and the Web, Department of Computer Science and Engineering, Chalmers University of Technology, University of Gothenburg, Gotemburgo, Suécia, 2015.

29. CUTLER, K. & CONSTINE J. - Facebook Buys Parse To Offer Mobile Development Tools As Its First Paid B2B Service - http://www.embarcados.com.br/modulo-esp8266/ Acesso em 11/05/2016

30. TEEUW, Michael - Parsing your IoT http://michaelteeuw.nl/post/116313960362/parsing-your-iot Acesso em 20/05/2016

31. BRAGHETTO, L. F. B & SILVA, S. C., Redes GSM e GPRS, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil, 2003.

32. MAGALHÃES, Paulo Sérgio (2003). Biometria e autenticação, Universidade do Minho, Guimarães, Portugal, 2003.

33. HORSTMANN, Cay, Conceitos de computação com o essencial de C++ - 3ł Edição: São José, Califórnia. San Jose State University, 2008.

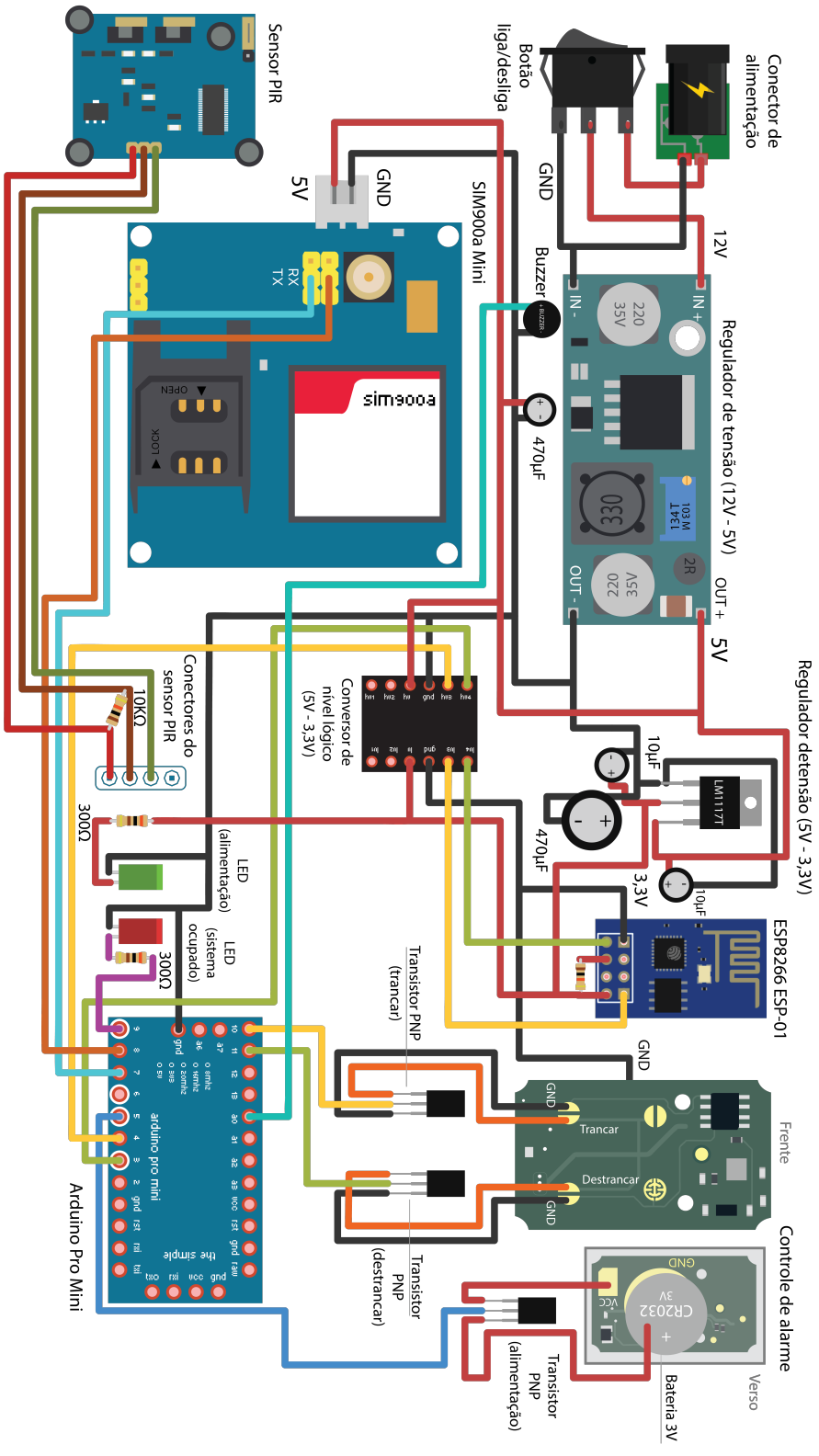
34. LAMB, Frank, Automação Industrial na Prática - Série Tekne. São Paulo: AMGH Editora, 2015.

35. SBORGI, D. & GIRIBONI, T., Monitoramento de sinais na indústria: Proposta de uma solução abrangendo hardware e software, II Encontro de Engenharia e Tecnologia dos Campos Gerais, Ponta Grossa, Paraná, Brasil, 2008.

36. LECHETA, Ricardo R., Desenvolvendo para iPhone e iPad 4ł edição, São Paulo. Novatec Editora Ltda., 2016.

Apêndice A - Esquemático da implementação física

Topologia do circuito com os componentes utilizados e citados neste trabalho. (Autor; Com algumas figuras da ferramenta Fritzing - `http://fritzing.org/`)



Apêndice B - Código da placa Arduino

Código de implementação das funcionalidades gerais pdara ser carregado na placa Arduino. Inclui lógica aplicada nos módulos e integração com a placa.

```
#include <SoftwareSerial.h>

#define BASE_DELAY 50 //Delay before executing next command on modules

//SoftwareSerial Module(RX, TX);
SoftwareSerial GPRS(7,8); //SIM900A Mini v3.8.2
SoftwareSerial ESP(3,4); //ESP8266 ESP-01

//Pins Variables
#define STATUS_LED_PIN 9 //On system init or working on request
#define ALARM_LOCK_PIN 10 //Alarm control lock button
#define ALARM_UNLOCK_PIN 11 //Alarm control unlock button
#define ALARM_VCC_PIN 5 //Turn Alarm On or Off
#define PIR_PIN 2 //PIR sensor connected pin
#define BUZZER_PIN 14 //Buzzer connected pin

//General Settings
#define VERBOSE 1 //Show details about operations
#define SEND_LOCATION_INTERVAL 1800000 //30 minutes (1800000 milliseconds)
unsigned long lastLocationUpdate = 0; //Last location update time (millis)
unsigned long lastStatusLedChange = 0; //Status LED blinking delay control (millis)
int lastStatusLedState = LOW; //Status LED blinking state control
boolean isSystemReady = false; //If system is busy working on a request or on init

/* ALARM CONTROL SETTINGS */
#define ALARM_ACTIVATION_DELAY 300 //Control button activation time


/* PIR SETTINGS */

boolean canDetectMotion = false; //If PIR is activated
boolean alreadyDetectedMotionAfterLock = false; //To avoid repeating alarms

/* GPRS SETTINGS */
#define CARD_PIN 5002 //SIM Card PIN Number


/* WEB SERVER SETTINGS */

//Address Parts to access service routes
#define SERVICE_PATH_VEHICLE_STATUS "/vStatus"
```

```
#define SERVICE_PATH_DEVICES "/macDevices"

//Parse Application ID Key used to grant permission to the service routes
#define SERVICE_PARSE_KEY "9Qv1SPg0W3jCz5bNlF7Ldny7w3yBYf1rIQlgvsr5"

//Parameter key for the Parse Application ID Key used on requests
#define SRV_P_PARSE_KEY "x-key"
//Keys of parameters used on vehicle status requests
#define SRV_P_LATITUDE "lat"
#define SRV_P_LONGITUDE "lng"
#define SRV_P_LOCKED_STATUS "isLocked"
#define SRV_P_MOMENT_ACTION "event"

//Response chars expected of some requests
#define SERVICE_SUCCESS_RESPONSE_EXPECTED "[DMOK]"

//Types of events - Sent with vehicle status
#define VEHICLE_STATUS_NO_ACTION 0
#define VEHICLE_STATUS_DID_LOCK 1
#define VEHICLE_STATUS_DID_UNLOCK 2
#define VEHICLE_STATUS_ALARM_FIRED 3

int moduleToListen = 0; //Which module Arduino should communicate

/* SERIAL SETTINGS */
#define SERIAL_BAUD_RATE 115200 //Rate used on Arduino
#define ESP_BAUD_RATE 57600 //Rate used on WiFi module (ESP8266)
#define GPRS_BAUD_RATE 57600 //Rate used on GPRS/GSM module (SIM900A)

/* ESP SETTINGS */
#define ESP_ID 100 //WiFi Module ID
#define ESP_CHECK_CLIENTS_INTERVAL 2000 //Show connected clients interval

//Command numbers used to activate some actions through app
#define APP_COMMAND_TOGGLE_ALARM 991 //Lock or unlock vehicle
#define APP_COMMAND_REQUEST_SYNC 992 //Request devices MAC addresses
#define APP_COMMAND_REQUEST_LOCATION 993 //Request to identify and send location

boolean espCheckClientsObserverActive = false; //Default State
String espSSID = "SSV"; //WiFi SSID
String espPassword = "20973560"; //WiFi Password
String authMacAddresses = "[DMaaaaaaaaaaaabbbbbbbbbbbbbDM]"; //Devices MACs
unsigned long lastClientsCheckMillis;
boolean authStatusBefore = false;
boolean authStatus = false;

//GPRS Settings
#define GPRS_ID 200 //GPRS Module ID
#define SERVER_REQUEST_URL "pf-20973560.parseapp.com"
```

```
int lastActionRun = 0; //Last GPRS Action (Group of Commands) was running
int lastCommandPart = 0; //Last GPRS Command (of the Last Action) was running
#define GPRS_COMMAND_INITIALIZE 1
#define GPRS_COMMAND_FIND_DEVICES 2
#define GPRS_COMMAND_SAVING_STATUS 3
int nextActionRunAfterInit = 0; //Next Action to run after GPRS initialize

//Last Location Register
String lastLatitude = "";
String lastLongitude = "";

//Last Vehicle Status Register
boolean lastIsLockedStatus = false; //Last lock state registered
int lastMomentAction = VEHICLE_STATUS_NO_ACTION; //Last event registered

void setup() {
  //Board pins modes
  pinMode(STATUS_LED_PIN, OUTPUT);
  pinMode(ALARM_VCC_PIN, OUTPUT);
  pinMode(ALARM_UNLOCK_PIN, OUTPUT);
  pinMode(ALARM_LOCK_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(PIR_PIN, INPUT);

  //Alarm control pins initial state
  digitalWrite(ALARM_VCC_PIN, HIGH);
  digitalWrite(ALARM_UNLOCK_PIN, HIGH);
  digitalWrite(ALARM_LOCK_PIN, HIGH);

  //Initialize Serial & GPRS for Communication
  Serial.begin(SERIAL_BAUD_RATE);
  while (!Serial) {;} //Wait for Arduino serial comm.
  ESP.begin(ESP_BAUD_RATE);
  while (!ESP) {;} //Wait for WiFi module serial comm.
  GPRS.begin(GPRS_BAUD_RATE);
  while (!GPRS) {;} //Wait for GPRS/GSM module serial comm.

  //Initialize WiFi Module
  initEspModule();

  //Wait for GPRS/GSM module to get ready
  listenToGprs();
  restartGprsModule(); //First commands to initialize
}

/*
  Handle GPRS initialization commands to
  avoid stopping in the middle of the process.
  If a command fails, it knows where to
```

```
  restart again using lastActionRun and
  lastCommandPart
*/
void nextGprsCommand() {
  if (lastActionRun == GPRS_COMMAND_INITIALIZE) {
    initGprsModule(lastCommandPart);
  } else if (lastActionRun == GPRS_COMMAND_FIND_DEVICES) {
    findDevicesFromCloud();
  } else if (lastActionRun == GPRS_COMMAND_SAVING_STATUS) {
    getCurrentLocationAndSaveLastVehicleStatusOnCloud();
  }
}

void loop() {
  //Communication with modules
  if (moduleToListen == ESP_ID) readEsp();
  if (moduleToListen == GPRS_ID) readGprs();
  readFromSerialWriteOnModules();

  //Request details of connected clients on WiFi module
  if (espCheckClientsObserverActive) checkClientsObserver();

  if (canDetectMotion) detectMotion(); //If PIR sensor can detect motion

  statusLedUpdate(); //Blinks if system us busy
  sendLastLocationIfNeeded(); //Send Location every {N} minutes
}

//Checks the interval since the last reporting location
void sendLastLocationIfNeeded () {
  if (isSystemReady) {
    if ((millis() - lastLocationUpdate) >= SEND_LOCATION_INTERVAL) {
      lastLocationUpdate = millis();
      lastMomentAction = VEHICLE_STATUS_NO_ACTION;
      getCurrentLocationAndSaveLastVehicleStatusOnCloud();
    }
  }
}

//Blinks LED status with certain delay when system is busy
void statusLedUpdate() {
  //When still loading
  if (!isSystemReady) {
    if (!lastStatusLedChange || (millis() - lastStatusLedChange >= 1000)) {
      lastStatusLedChange = millis();
      lastStatusLedState = !lastStatusLedState;
      digitalWrite(STATUS_LED_PIN, lastStatusLedState);
    }
  }
```

```
}

//Sets if system is ready or busy
void systemIsReady(boolean ready) {
  isSystemReady = ready; //Set ready state
  digitalWrite(STATUS_LED_PIN, !ready); //Turn status LED on or off
}

//Checks PIR sensor state
void detectMotion() {
  //If not detected any motion after vehicle lock command, can get sensor value
  if (alreadyDetectedMotionAfterLock == false) {
    boolean hasMotion = digitalRead(PIR_PIN);
    if (hasMotion && lastIsLockedStatus) { //If Detected Motion & Car is Locked
      didDetectMotion();
    }
  }
}

//Signal tone to notify system events
void buzzerBeep (int f) {
  tone(BUZZER_PIN,f);
  delay(100);
  noTone(BUZZER_PIN);
}

//When sensor detects motion
void didDetectMotion() {
  if (VERBOSE) Serial.println("[PIR]");
  buzzerBeep(2000);
  buzzerBeep(1600);
  buzzerBeep(1200);
  buzzerBeep(800);
  alreadyDetectedMotionAfterLock = true;
  lastMomentAction = VEHICLE_STATUS_ALARM_FIRED;
  getCurrentLocationAndSaveLastVehicleStatusOnCloud();
}

//Request list of connected clients on WiFi module
void checkClientsObserver() {
  if ((millis() - lastClientsCheckMillis) >= ESP_CHECK_CLIENTS_INTERVAL) { //3 Seconds Timer
    espCheckClients();
    if (authStatus != authStatusBefore) { //Status Changed
      if (authStatus) {
        //Alarm Control - Unlock Command
        unlockVehicle();
        //
      } else {
        //Alarm Control - Lock Command
```

```
        lockVehicle();
      }
      //Send Information to Cloud
      getCurrentLocationAndSaveLastVehicleStatusOnCloud();
      authStatusBefore = authStatus;
    }
  }
}

//Activates alarm control to lock vehicle
void lockVehicle () {
  digitalWrite(ALARM_LOCK_PIN, LOW);
  delay(ALARM_ACTIVATION_DELAY);
  digitalWrite(ALARM_LOCK_PIN, HIGH);
  lastMomentAction = VEHICLE_STATUS_DID_LOCK; //Locked Event
  lastIsLockedStatus = true; //Locked Vehicle
  alreadyDetectedMotionAfterLock = false;
  buzzerBeep(600);
}

//Activates alarm control to unlock vehicle
void unlockVehicle () {
  digitalWrite(ALARM_UNLOCK_PIN, LOW);
  delay(ALARM_ACTIVATION_DELAY);
  digitalWrite(ALARM_UNLOCK_PIN, HIGH);
  lastMomentAction = VEHICLE_STATUS_DID_UNLOCK; //Unlocked Event
  lastIsLockedStatus = false; //Unlocked Vehicle
  buzzerBeep(1600);
  buzzerBeep(1600);
}

//Sends command to list connected clients to WiFi module
void espCheckClients() {
  systemIsReady(true);
  lastClientsCheckMillis = millis();
  listenToEsp();
  //Check Clients Connected to AP
  ESP.println("AT+CWLIF");
  readEsp();
}

//Analyze data received from WiFi module
void readEsp() {
  ESP.listen();

  while(ESP.available()) {
    String line = ESP.readStringUntil('\n');
    Serial.println("ESP: " + line);
    //Check any command after start of line
```

```
    String command = line.substring(line.indexOf("ESP: ")+1);
    //Check if received client information like IP and MAC Addresses

    //Received Command from App
    if (command.startsWith("+IPD") && authStatus) { //If Authorized
      if (ESP.find("comm")) {
        int commCode = ESP.parseInt();
        if (commCode == APP_COMMAND_TOGGLE_ALARM) { //Toggle Alarm Control
          if (lastIsLockedStatus) {
            unlockVehicle();
          } else {
            lockVehicle();
          }
        } else if (commCode == APP_COMMAND_REQUEST_SYNC) { //Update Authorized Devices
          findDevicesFromCloud();
        } else if (commCode == APP_COMMAND_REQUEST_LOCATION) { //Update Location
          lastMomentAction = VEHICLE_STATUS_NO_ACTION;
          getCurrentLocationAndSaveLastVehicleStatusOnCloud();
        }
      }
    }

    if (command.startsWith("AT+CWLIF") || command.startsWith("192.")) { //Had response of
        CWLIF command
      authStatus = false;
      int indexOfMac = command.indexOf(",")+1;
      String mac = command.substring(indexOfMac,indexOfMac+17); //Avoid line break
      if (mac.length() >= 17) {
        removeDots(mac); //Remove all ':'
        int indexOfFoundAuth = authMacAddresses.indexOf(mac); //Try to found
        if (indexOfFoundAuth > -1) { //If Found
          authStatus = true;
        }
      }
    }

    // if (ESP.find("[TOGGLE_ALARM]")) {
    //   Serial.println("CAN TOGGLE ALARM\n");
    // }
  }
}

//Remove non-numeric chars from MAC addresses
void removeDots (String &address) {
  int indexOfDots = address.indexOf(':');
  if (indexOfDots > 0) {
    address.remove(indexOfDots,1);
    removeDots(address);
  }
```

```
}

//WiFi module setup commands
void initEspModule() {
  /*
    These initialization commands setup any ESP8266 module
    to work in AP mode and update the SSID and password
    if needed
  */
  //Set WiFi mode as access point mode
  ESP.println("AT+CWMODE=2");
  delay(BASE_DELAY);
  //Update WiFi SSID and Password
  ESP.println("AT+CWSAP=\"" + espSSID + "\"," + "\"" + espPassword + "\",9,2");
  delay(BASE_DELAY);
  //Multiple Connections
  ESP.println("AT+CIPMUX=1");
  delay(BASE_DELAY);
  //Create Server - Port 80
  ESP.println("AT+CIPSERVER=1,80");
  delay(BASE_DELAY);
  //Two Seconds Timeout
  ESP.println("AT+CIPSTO=1");
  delay(BASE_DELAY);
}

//Set PIN of SIM card, when it has
void setGprsPinCard() {
    //Set Card PIN if needed
  if (String(CARD_PIN).length()) {
    GPRS.println("AT+CPIN=\"" + String(CARD_PIN) + "\"");
    delay(BASE_DELAY);
  }
}

//Restarts GPRS/GSM module
void restartGprsModule() {
  GPRS.println("AT+CFUN=1,1");
  delay(BASE_DELAY);
  readGprs();
  GPRS.println("AT+CMEE=2");
  delay(BASE_DELAY);
  readGprs();
}

//Setup commands to initialize GPRS comm.
void initGprsModule(int part) {
  //Register last command
  lastActionRun = GPRS_COMMAND_INITIALIZE;
```

```
  systemIsReady(false); //System is busy

  if (part == 1 || !part) {
    //Setting the SAPBR, the connection type is using GPRS
    lastCommandPart = 1;
    GPRS.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");
    delay(BASE_DELAY);
    readGprs();
    if (part == 1) return;
  }

  if (part == 2 || !part) {
    //Setting the APN, the second need you fill in your local APN server
    lastCommandPart = 2;
    GPRS.println("AT+SAPBR=3,1,\"APN\",\"tim.br\"");
    delay(BASE_DELAY);
    readGprs();
    if (part == 2) return;
  }

  if (part == 3 || !part) {
    //Setting the SAPBR, for detail you can refer to the AT command manual
    lastCommandPart = 3;
    GPRS.println("AT+SAPBR=1,1");
    delay(BASE_DELAY);
    readGprs();
    if (part == 3) return;
  }

  if (part == 4) {
    //Check next step
    if (nextActionRunAfterInit) {
      lastActionRun = nextActionRunAfterInit; //Go to programmed
    } else {
      lastActionRun++; //Go to next index
    }
    lastCommandPart = 0;
    nextGprsCommand();
  }
}

//Sends command to GPRS/GSM module to identify current location
void getCurrentLocationAndSaveLastVehicleStatusOnCloud () {
  buzzerBeep(1800);
  buzzerBeep(800);
  buzzerBeep(1800);

  espCheckClientsObserverActive = false; //Stop checking clients for a while
```

```
  canDetectMotion = false; //Stop detecting motion for a while
  nextActionRunAfterInit = GPRS_COMMAND_SAVING_STATUS; //Force to retry it if fails
  listenToGprs();
  GPRS.println("AT+CIPGSMLOC=1,1");
  readGprs();
}

//HTTP initialization commands
void initializeHttp (String path) {
  systemIsReady(false);

  //Init the HTTP request
  GPRS.println("AT+HTTPINIT");
  delay(BASE_DELAY);
  readGprs();

  //Setting the URL HTTP Parameter
  GPRS.println("AT+HTTPPARA=\"URL\",\"" + String(SERVER_REQUEST_URL) + path + "\"");
  delay(BASE_DELAY);
  readGprs();

  //Setting the Correlation ID HTTP Parameter
  GPRS.println("AT+HTTPPARA=\"CID\",1");
  delay(BASE_DELAY);
  readGprs();

  //Settings the Content-Type HTTP Header Parameter
  GPRS.println("AT+HTTPPARA=\"CONTENT\",\"application/json\"");
  delay(BASE_DELAY);
  readGprs();
}

//Build JSON body to use on post request
void submitRequestWithBody (String jsonBody) {
  //Inputting HTTP Data with Supported Body (JSON) Size and Supported Time
  GPRS.println("AT+HTTPDATA=" + String(jsonBody.length()) + ",6000");
  delay(BASE_DELAY);
  readGprs();

  //Input Data
  GPRS.println(jsonBody); //init the HTTP request
  delay(BASE_DELAY);
  readGprs();

  //Submit the POST Request and Wait for the Response
  GPRS.println("AT+HTTPACTION=1");
  delay(BASE_DELAY);
  readGprs();
}
```

```
//Called when a response is received
void showRequestResponse() {
  //Show Response Received
  if (VERBOSE) {
    GPRS.println("AT+HTTPREAD");
    delay(BASE_DELAY);
    readGprs();
  }
}

//Termination commands of HTTP requests
void terminateHttpService() {
  //Close the HTTP Service
  GPRS.println("AT+HTTPTERM");
  delay(BASE_DELAY);
  readGprs();
  systemIsReady(true); //Ready
}

//Requests cloud service to retrieve MAC addresses of authorized devices
void findDevicesFromCloud() {
  buzzerBeep(2000);
  buzzerBeep(2200);
  buzzerBeep(2400);

  lastActionRun = GPRS_COMMAND_FIND_DEVICES;

  initializeHttp(SERVICE_PATH_DEVICES);

  //Parse Key & Vehicle Status in JSON Format
  String json = "";
  json += "{";
  json += "\"" + String(SRV_P_PARSE_KEY) + "\":\"" + String(SERVICE_PARSE_KEY) + "\"";
  json += "}";

  submitRequestWithBody(json);
}

//Requests cloud to save current vehicle status
void saveLastVehicleStatusOnCloud () {
  initializeHttp(SERVICE_PATH_VEHICLE_STATUS);

  //Parse Key & Vehicle Status in JSON Format
  String json = "";
  json += "{";
  json += "\"" + String(SRV_P_PARSE_KEY) + "\":\"" + String(SERVICE_PARSE_KEY) + "\",";
  json += "\"" + String(SRV_P_LOCKED_STATUS) + "\" :" + String(lastIsLockedStatus) + ",";
  json += "\"" + String(SRV_P_LATITUDE) + "\":\"" + lastLatitude + "\",";
```

```
  json += "\"" + String(SRV_P_LONGITUDE) + "\":\"" + lastLongitude + "\",";
  json += "\"" + String(SRV_P_MOMENT_ACTION) + "\":" + String(lastMomentAction);
  json += "}";

  submitRequestWithBody(json);
}

//Analyze data received from GPRS/GSM module
void readGprs() {
  GPRS.listen();
  while(GPRS.available()) {
    //Get entire line & show if verbose of commands is on
    String line = GPRS.readStringUntil('\r\n');

    //Check any command after start of line
    String command = line.substring(line.indexOf(":")+1);

    //Expected commands and its actions here
    Serial.println("GPRS: " + line);

    //Detect Module Initialization - Set PIN
    if (line.startsWith("+CPIN: SIM PIN") ||
      line.startsWith("+CME ERROR: SIM PIN")) {
      setGprsPinCard(); //Set GPRS Sim Card
    }

    //Detect Module Initialization - After PIN Set
    if (line.startsWith("Call Ready")) {
      if (VERBOSE) Serial.println("[GPRS RDY]");
      initGprsModule(0);
    }

    if (line.startsWith("OK") && lastActionRun == GPRS_COMMAND_INITIALIZE) {
      lastCommandPart++;
      nextGprsCommand();
    }

    //Detected GPRS Command Error - Retry the Command
    //Occurs mainly when trying to connect to network
    if (line.startsWith("+CME ERROR:")) {
      if (lastActionRun == GPRS_COMMAND_INITIALIZE) {
        delay(4000); //Wait
        nextGprsCommand(); //Retry
      }
    }

    //Detected GPRS Command Error - Retry the Command
    //Probably, a network error when connecting
    if (line.startsWith("ERROR")) {
```

```
    if (lastActionRun == GPRS_COMMAND_INITIALIZE) {
      delay(4000); //Wait
      nextGprsCommand(); //Retry
    }
  }

  //Check if received Location Response, refresh variables
  if (line.startsWith("+CIPGSMLOC: ")) {
    String tmp = line.substring(line.indexOf(",")+1);
    //Validate (Ex.: '+CIPGSMLOC: 0,-47.888781,-15.752166,2016/03/26,03:10:31' is more
        than 20 chars)
    if (tmp.length() > 20) {
      lastLongitude = tmp.substring(0,tmp.indexOf(","));
      tmp = tmp.substring(tmp.indexOf(",")+1);
      lastLatitude = tmp.substring(0,tmp.indexOf(","));
      if (VERBOSE) Serial.println("[SAVE LOC]");
      saveLastVehicleStatusOnCloud(); //Save on Cloud
    } else { //Potential Error
      if (line.indexOf("60")) { //Network error - starts with 6
        //Restart
        espCheckClientsObserverActive = false;
        listenToGprs();
        restartGprsModule();
      }
    }
  } else if (line.startsWith("+HTTPACTION:")) { //Check if any HTTP response was received
    if (VERBOSE) Serial.println("[GOT RSP]"); //Got Response
    showRequestResponse();
  }

  //If Received Success Response Expected
  if (line.indexOf(String(SERVICE_SUCCESS_RESPONSE_EXPECTED)) > 0) {
    if (VERBOSE) Serial.println("[SAVED]");
    espCheckClientsObserverActive = true; //Continue to check clients
    canDetectMotion = true;
  }

  //If Received Success Response Finding Devices
  int devicesMacsStart = line.indexOf("[DM");
  int devicesMacsEnd = line.indexOf("DM]");
  if ((devicesMacsStart == 0) && (devicesMacsEnd > 0)) { //Found
    authMacAddresses = line.substring(devicesMacsStart+3, devicesMacsEnd);
    if (VERBOSE) Serial.println("[MACSOK]");
    espCheckClientsObserverActive = true;
    canDetectMotion = true;
    terminateHttpService();
  }

  //Received 601 (Network error) or 603 (DNS Error) or related
```

```
    if (line.startsWith("+HTTPACTION:1,60")) {
      if (lastActionRun == GPRS_COMMAND_SAVING_STATUS) {
        delay(4000); //Wait
        nextGprsCommand(); //Retry
      }
    }

  }
}

//Starts listening to WiFi module
void listenToEsp() {
  if (moduleToListen != ESP_ID) {
    if (VERBOSE) Serial.println("[ESP LSTN]");
    moduleToListen = ESP_ID;
  }
}

//Starts listening to GPRS/GSM module
void listenToGprs() {
  if (moduleToListen != GPRS_ID) {
    if (VERBOSE) Serial.println("[GPRS LSTN]");
    moduleToListen = GPRS_ID;
  }
}

//Write Data on GPRS inputed from Serial
void readFromSerialWriteOnModules() {
  while (Serial.available()) {
    String line = Serial.readStringUntil('\n');
    if (line.startsWith("GPRS:")) {
      listenToGprs();
      String lineWithoutStart = line.substring(line.indexOf(":")+1);
      GPRS.println(lineWithoutStart);
    } else if (line.startsWith("ESP:")) {
      listenToEsp();
      String lineWithoutStart = line.substring(line.indexOf(":")+1);
      ESP.println(lineWithoutStart);
    }
  }
}
```

Apêndice C - Código utilizado no serviço em nuvem

Código utilizado no 'Parse Cloud Code', serviço MBaaS utilizado no projeto. Inclui envio de notificações, criação de rotas do serviço, ações de consulta e registros. Uso de algumas técnicas utilizadas disponíveis em `http://michaelteeuw.nl/post/116313960362/parsing-your-iot`. Acesso em 20/05/2016. (TEEUW, http://michaelteeuw.nl, 2016).

```
require('cloud/app.js');
```

```
//Initialize Express in Cloud Code
express = require('express');
app = express();

//Required API Key
var apiKey = '9Qv1SPg0W3jCz5bNlF7Ldny7w3yBYf1rIQlgvsr5';

//Global app configuration section
app.use(express.bodyParser()); //Middleware for reading request body

//To Receive MAC Addresses of All Authorized Devices
app.post('/macDevices', function(req, res) {
  //Body parameter received
  var requestApiKey = req.body['x-key'];

  //Check the API key
  if (requestApiKey === undefined || requestApiKey !== apiKey) {
    res.status(401).send({error:"Missing or incorrect API-key."});
    return;
  };

  //Query
  var Device = Parse.Object.extend('Device');
  var query = new Parse.Query(Device);
  //Get all devices saved
  query.find({
   success: function(results) {
    //Register on log
     alert("Successfully retrieved " + results.length + " devices.");
     //Retrieved Devices - Check Length:
      if (results.length) { //If there's any device
        var macs = [];
        // Do something with the returned Parse.Object values
        for (var i = 0; i < results.length; i++) {
```

```
            var object = results[i];
            var macAddress = object.get('macAddress');
            macs.push(macAddress);
          }
          //[DM - Devices MAC Addresses Start
          //DM] - Devices MAC Addresses End
          var macsString = macs.toString();
          //Remove any chars but numbers
          macsString = macsString.split(":").join("");
          res.send('[DM' + macsString + 'DM]');
          //Send silent push to notify user that list has veen synchronized
          sendPush("Sistema inicializado. " +
          "A lista de dispositivos foi sincronizada.", 0);
        } else { //List has no devices
          //Send silent push to notify user that there's no devices
          sendPush("Nao ha nenhum dispositivo autorizado. " +
          "Adicione um dispositivo.", 0);
          res.send({r: '[DM-DM]'});
        }
      },
      error: function(error) {
       alert("Error: " + error.code + " " + error.message);
      }
    });
});

//To Send Vehicle Status
app.post('/vStatus', function(req, res) {
  //Body parameters received
  var requestApiKey = req.body['x-key'];
  var isLocked = req.body.isLocked;
  var event = req.body.event;
  var latitude = req.body.lat;
  var longitude = req.body.lng;

  //Check the API key
  if (requestApiKey === undefined || requestApiKey !== apiKey) {
    res.status(401).send({error:"Missing or incorrect API-key."});
    return;
  };

  //Validate Locked Status
  if (isLocked === undefined) {
    res.status(400).send({error:"isLocked missing."});
    return;
  } else {
    isLocked = Boolean(isLocked);
    if (typeof isLocked !== "boolean") {
      res.status(400).send({error:"isLocked is not a boolean."});
```

```
      return;
    }
};

//Validate Moment Action
if (event === undefined) {
    res.status(400).send({error:"Event missing."});
    return;
} else if (typeof event !== "number") {
    res.status(400).send({error:"Event is not a number."});
    return;
};

//Validate Latitude
if (latitude === undefined) {
    res.status(400).send({error:"Latitude missing."});
    return;
} else if (latitude.length <= 0) {
    res.status(400).send({error:"Latitude sent without length."});
    return;
} else if (typeof latitude !== "string") {
    res.status(400).send({error:"Latitude is not a valid string."});
    return;
};

//Validate Longitude
if (longitude === undefined) {
    res.status(400).send({error:"Longitude missing."});
    return;
} else if (longitude.length <= 0) {
    res.status(400).send({error:"Longitude sent without length."});
    return;
} else if (typeof longitude !== "string") {
    res.status(400).send({error:"Longitude is not a valid string."});
    return;
};

//Create the Vehicle Status object.
var VehicleStatus = Parse.Object.extend("VehicleStatus");
var newVehicleStatus = new VehicleStatus();
newVehicleStatus.set('isLocked', isLocked);
newVehicleStatus.set('event', event);
newVehicleStatus.set('latitude',latitude);
newVehicleStatus.set('longitude', longitude);
//Save object
newVehicleStatus.save(null, {
    success: function (newVehicleStatus) {
      //Everything ok. report error to client.
      res.send({"r": "[DMOK]"});
```

```
    },
    error: function (error) {
      //Error saving the log. Report error to client.
      res.status(400).send({error:"Log save error: " +
                              error.code + " " +
                              error.message});
    }
  });
});

//Send push to global channel - all devices with app installed
function sendPush (msg, type) {
  var sound;
  if (type == 0) sound = ""; //No Sound
  if (type == 1) sound = "default"; //Normal Sound
  if (type == 2) sound = "alarm.wav"; //Alarm Sound
  Parse.Push.send({
    channels: [ "global" ],
    data: {
      sound: sound,
      alert: msg
    }
  }, {
    success: function() {
      //Log: Push was successful
      console.log("Push notification sent to all devices.");
    },
    error: function(error) {
      //Log: Handle error
      console.log("Error trying to send push notification to all devices.")
    }
  });
}

//After Saving Vehicle Status, Send Notification
Parse.Cloud.afterSave("VehicleStatus", function(request) {
  //Get the VehicleStatus object
  var query = new Parse.Query("VehicleStatus");
  query.equalTo("objectId", request.object.id);
  query.find({
   success: function(results) {
      var vehicleStatus = results[0];
      if (vehicleStatus) {
        //Use event number to get it's description
        var eventNumber = vehicleStatus.get("event");
        var Event = Parse.Object.extend('Event');
        var query = new Parse.Query(Event);
        query.equalTo("eventNumber", eventNumber);
        query.find({
```

```
          success: function(results) {
              var event = results[0];
              if (event) {
                //Setting sound default sound
                var soundType = 1;
                var eventDescription = event.get("pushDescription");
                var isLocked = vehicleStatus.get("isLocked");
                var alertMsg = eventDescription; //Got from Event table
                //Describe Locked Status if event is not relevant
                if (eventNumber == 0 || eventNumber == 3) {
                  if (isLocked) {
                    alertMsg += "Seu veiculo esta trancado no momento.";
                  } else {
                    alertMsg += "Seu veiculo esta destrancado no momento.";
                  }
                }
                //Set sound as alarm if event type is 3
                if (eventNumber == 3) soundType = 2;
                sendPush(alertMsg, soundType);
              }
          },
          //Didn't find event with given number
          error: function(error) {
            console.error("Error finding event object with number " +
              eventNumber + error.code + ": " + error.message);
          }
        });
      }
   },
   //Didn't find recently saved object
   error: function(error) {
     console.error("Error finding the recent included Vehicle Status Object "
       + error.code + ": " + error.message);
   }
  });
});

//Enable the express webserver.
app.listen();
```

Apêndice D - Código do aplicativo em iOS

Arquivos de código de implementação do aplicativo na plataforma iOS.

Arquivo de configuração da ferramenta de gerenciamento de dependências Cocoapods - Podfile:

```
# Uncomment this line to define a global platform for your project
# platform :ios, '8.0'
# Uncomment this line if you're using Swift
# use_frameworks!

target 'SSV Mobile' do

end

use_frameworks!

pod 'Parse'
pod 'SystemServices'
```

Arquivo tipo 'interface' da tela de adicionar dispositivo autorizado - AddDeviceTableViewController.h:

```
//
// AddDeviceTableViewController.h
// SSV Mobile
//
// Created by Allan Alves on 4/6/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface AddDeviceTableViewController : UITableViewController

@end
```

Arquivo tipo 'implementation' da tela de adicionar dispositivo autorizado - AddDeviceTableViewController.m:

```
//
// AddDeviceTableViewController.m
// SSV Mobile
```

```
//
// Created by Allan Alves on 4/6/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "AddDeviceTableViewController.h"

@interface AddDeviceTableViewController ()

@property (weak, nonatomic) IBOutlet UIBarButtonItem *doneButton;

@property (weak, nonatomic) IBOutlet UITextField *deviceNameTextField;
@property (weak, nonatomic) IBOutlet UITextField *deviceModelTextField;
@property (weak, nonatomic) IBOutlet UITextField *deviceMacTextField;

@end

@implementation AddDeviceTableViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}



- (void)viewWillAppear:(BOOL)animated {
   [super viewWillAppear:animated];

   //Device Details
   NSString *deviceSize;
   if (IS_IPAD) deviceSize = @"iPad";
   if (IS_IPHONE_4_OR_LESS) deviceSize = @"iPhone 4";
   if (IS_IPHONE_5) deviceSize = @"iPhone 5 (ou 5S)";
   if (IS_IPHONE_6) deviceSize = @"iPhone 6 (ou 6S)";
   if (IS_IPHONE_6P) deviceSize = @"iPhone 6 Plus (ou 6S Plus)";

   _deviceNameTextField.text = [[UIDevice currentDevice] name];
   _deviceModelTextField.text = deviceSize;
   [_deviceMacTextField becomeFirstResponder];
}

- (IBAction)getMacAddress:(id)sender {
   UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"çEndereo MAC"
                                                                   message:@"Copie o çendereo MAC Wi-Fi em
                                                                       Ajustes > Geral > Sobre > Wi-Fi e cole
                                                                       no campo de texto \"çEndereo MAC\""
                                                                       preferredStyle:UIAlertControllerStyleAlert];
   UIAlertAction *settingsAction = [UIAlertAction actionWithTitle:@"Ir para Ajustes"
       style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
```

```
      [[UIApplication sharedApplication] openURL:[NSURL
          URLWithString:@"prefs:root=General&path=About&Path=WIFI"]];
   }];

   UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancelar"
       style:UIAlertActionStyleCancel handler:nil];

   [alert addAction:settingsAction];
   [alert addAction:cancelAction];

   [self presentViewController:alert animated:YES completion:nil];
}

- (void)viewDidAppear:(BOOL)animated {
   [super viewDidAppear:animated];
}

#pragma mark - Actions

- (IBAction)typringTextField:(id)sender {
   BOOL allValid = true;
   if (_deviceNameTextField.text.length < 3) {
      allValid = false;
   }
   if (_deviceModelTextField.text.length < 3) {
      allValid = false;
   }
   if (_deviceMacTextField.text.length < 17) {
      allValid = false;
   }
   [_doneButton setEnabled:allValid];
}

- (IBAction)doneAddingDevice:(id)sender {
   PFObject *newDevice = [PFObject objectWithClassName:kParseTableDevice];
   [newDevice setValue:_deviceNameTextField.text forKey:kParseColumnDeviceName];
   [newDevice setValue:_deviceModelTextField.text forKey:kParseColumnDeviceModel];
   [newDevice setValue:[_deviceMacTextField.text lowercaseString]
       forKey:kParseColumnDeviceMacAddress];

   [_doneButton setEnabled:NO];
   [newDevice saveInBackgroundWithBlock:^(BOOL succeeded, NSError * _Nullable error) {
      if (succeeded) {
         [self.navigationController popViewControllerAnimated:YES];
      }
   }];
}

@end
```

Arquivo tipo ’interface’ do protocolo ’Application Delegate’ - AppDelegate.h:

```
//
// AppDelegate.h
// SSV Mobile
//
// Created by Allan on 3/23/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <Parse/Parse.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

Arquivo tipo ’implementation’ protocolo ’Application Delegate’ - AppDelegate.m:

```
//
// AppDelegate.m
// SSV Mobile
//
// Created by Allan on 3/23/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
    *)launchOptions {

   //Initialize Parse
   [Parse setApplicationId:PARSE_APP_ID clientKey:PARSE_CLIENT_KEY];

   //Set Global Appearance

   [self registerForRemoteNotificationsWithApplication:application];

   //Get All Events
```

```
   [General requestAllEventsWithCompletion:^(BOOL success) {
      [[NSNotificationCenter defaultCenter] postNotificationName:@"AllEventsLoaded"
          object:nil];
   }];

   return YES;
}

- (void) registerForRemoteNotificationsWithApplication:(UIApplication*)application {
   UIUserNotificationType userNotificationTypes = (UIUserNotificationTypeAlert |
                                       UIUserNotificationTypeBadge |
                                       UIUserNotificationTypeSound);
   UIUserNotificationSettings *settings = [UIUserNotificationSettings
       settingsForTypes:userNotificationTypes
                                                             categories:nil];
   [application registerUserNotificationSettings:settings];
   [application registerForRemoteNotifications];
}

- (void)application:(UIApplication *)application
    didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
   // Store the deviceToken in the current installation and save it to Parse.
   PFInstallation *currentInstallation = [PFInstallation currentInstallation];
   [currentInstallation setDeviceTokenFromData:deviceToken];
   currentInstallation.channels = @[ @"global" ];
   [currentInstallation saveInBackground];
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
    *)userInfo {
   [PFPush handlePush:userInfo];
}

@end
```

Arquivo tipo ’interface’ da tela de acionar comandos no dispositivo - ControlTable-ViewController.h:

```
//
// ControlTableViewController.h
// SSV Mobile
//
// Created by Allan Alves on 6/8/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <SystemServices/SystemServices.h>
```

```
@interface ControlTableViewController : UITableViewController <NSURLConnectionDelegate>

@end
```

Arquivo tipo 'implementation' da tela de acionar comandos no dispositivo - ControlTableViewController.m:

```
//
// ControlTableViewController.m
// SSV Mobile
//
// Created by Allan Alves on 6/8/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "ControlTableViewController.h"

typedef enum {
   ControlSectionAlarm = 0,
   ControlSectionDevicesList = 1,
   ControlSectionUpdateLocation = 2
}ControlSection;

@interface ControlTableViewController ()

@end

@implementation ControlTableViewController {
   NSString *wifiRouterIP;
}

- (void)viewDidAppear:(BOOL)animated {
   [super viewDidAppear:animated];
}

#pragma mark - Table View Delegate

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

   if (indexPath.section == ControlSectionAlarm) {
      //Lock & Unlock
      [self sendRequestWithCommand:@"991"]; //Toggle Alarm Control
   } else if (indexPath.section == ControlSectionDevicesList) {
      //Sincronize Devices List
      [self sendRequestWithCommand:@"992"]; //Update Authorized Devices
   } else if (indexPath.section == ControlSectionUpdateLocation) {
      //Update Location and Save Status
```

```
        [self sendRequestWithCommand:@"993"]; //Update Location
    }

    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}

- (void) updateWifiIP {
    wifiRouterIP = [[SystemServices sharedServices] wiFiIPAddress];
    NSArray *components = [wifiRouterIP componentsSeparatedByString:@"."];
    //Set Router IP
    wifiRouterIP = [NSString stringWithFormat:@"%@.%@.%@.1", components[0], components[1],
        components[2]];
}

#pragma mark - Actions

- (void) sendRequestWithCommand:(NSString*)command {
    [self updateWifiIP];

    NSString *url = [NSString stringWithFormat:@"http://%@:80", wifiRouterIP];

    NSString *post = [NSString stringWithFormat:@"comm=%@", command];
    NSData *postData = [post dataUsingEncoding:NSUTF8StringEncoding];

    NSMutableURLRequest *request = [[NSMutableURLRequest alloc] init];
    [request setURL:[NSURL URLWithString:url]];

    [request setHTTPMethod:@"POST"];

    [request setHTTPBody:postData];

    NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:request
        delegate:self];

    if(connection) {
        NSLog(@"Connected");
    } else {
        NSLog(@"Not Connected");
    }
}

#pragma mark - URL Connection Delegate

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData*)data {
    NSLog(@"%@", [NSString stringWithUTF8String:[data bytes]]);
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
    NSLog(@"%@", error);
```

```
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
   NSLog(@"%@", connection);
}

@end
```

Arquivo tipo 'interface' da classe herdada de UITextField para uso de máscara para endereço MAC - CustomTextField.h:

```
//
// CustomTextField.h
// detranMovel
//
// Created by Allan on 11/24/15.
// Copyright (c) 2015 Infosolo. All rights reserved.
//

#import <UIKit/UIKit.h>

#define TextFieldMaskTypeMacAddress @"TextFieldMaskTypeMacAddress"

@interface CustomTextField : UITextField <UITextFieldDelegate>

@property (nonatomic) IBInspectable NSInteger maxLength;
@property (nonatomic) IBInspectable NSString *maskType;

@end
```

Arquivo tipo 'implementation' da classe herdada de UITextField para uso de máscara para endereço MAC - CustomTextField.m:

```
//
// CustomTextField.m
// detranMovel
//
// Created by Allan on 11/24/15.
// Copyright (c) 2015 Infosolo. All rights reserved.
//

#import "CustomTextField.h"

@implementation CustomTextField

- (void)awakeFromNib {
   [super awakeFromNib];
   self.delegate = self;
```

```
}

- (BOOL)textField:(UITextField *) textField shouldChangeCharactersInRange:(NSRange)range
    replacementString:(NSString *)string {

   NSString * appendString = @"";
   //Mac - 00:00:00:00:00:00
   if ([self.maskType isEqualToString:TextFieldMaskTypeMacAddress]){ //Mac text field
      if (!range.length) {
         switch (range.location) {
            case 2:
            case 5:
            case 8:
            case 11:
            case 14:
               appendString = @":";
               break;
            default:
               break;
         }
      }
   }

   NSString *newString = [textField.text stringByAppendingString:appendString];

   [textField setText:newString];

   //Ignore if max length is zero
   if (self.maxLength == 0) {
      return true;
   }

   NSUInteger oldLength = [textField.text length];
   NSUInteger replacementLength = [string length];
   NSUInteger rangeLength = range.length;

   NSUInteger newLength = oldLength - rangeLength + replacementLength;

   BOOL returnKey = [string rangeOfString: @"\n"].location != NSNotFound;

   return newLength <= self.maxLength || returnKey;
}

@end
```

Arquivo tipo ’interface’ da tela de detalhes de um dispositivo autorizado - DeviceDetailsTableViewController.h:

```
//
// DeviceDetailsTableViewController.h
// SSV Mobile
//
// Created by Allan Alves on 4/6/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface DeviceDetailsTableViewController : UITableViewController

@property (nonatomic) PFObject *selectedDevice;

@end
```

Arquivo tipo ’implementation’ da tela de detalhes de um dispositivo autorizado - DeviceDetailsTableViewController.m:

```
//
// DeviceDetailsTableViewController.m
// SSV Mobile
//
// Created by Allan Alves on 4/6/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "DeviceDetailsTableViewController.h"

@interface DeviceDetailsTableViewController ()

@property (weak, nonatomic) IBOutlet UILabel *deviceModelLabel;
@property (weak, nonatomic) IBOutlet UILabel *deviceNameLabel;
@property (weak, nonatomic) IBOutlet UILabel *deviceMacLabel;

@end

@implementation DeviceDetailsTableViewController

- (void)viewDidLoad {
   [super viewDidLoad];

   if (_selectedDevice) {
      _deviceNameLabel.text = [_selectedDevice objectForKey:kParseColumnDeviceName];
      _deviceModelLabel.text = [_selectedDevice objectForKey:kParseColumnDeviceModel];
      _deviceMacLabel.text = [_selectedDevice objectForKey:kParseColumnDeviceMacAddress];
   }
}
```

```
#pragma mark - Actions

- (IBAction)removeDevice:(id)sender {
   [_selectedDevice deleteInBackgroundWithBlock:^(BOOL succeeded, NSError * _Nullable error) {
      if (succeeded) {
         [General requestAllDevicesWithCompletion:^(BOOL success) {
            if (success) {
               [self.navigationController popViewControllerAnimated:YES];
            }
         }];
      }
   }];
}

@end
```

Arquivo tipo ’interface’ da tela de dispositivos autorizados - DevicesTableViewController.h:

```
//
// DevicesTableViewController.h
// SSV Mobile
//
// Created by Allan on 3/24/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface DevicesTableViewController : UITableViewController

@end
```

Arquivo tipo ’implementation’ da tela de dispositivos autorizados - DevicesTableViewController.m:

```
//
// DevicesTableViewController.m
// SSV Mobile
//
// Created by Allan on 3/24/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "DevicesTableViewController.h"
#import "DeviceDetailsTableViewController.h"
```

```
@interface DevicesTableViewController ()

@property (strong, nonatomic) IBOutlet UIView *loadingView;
@property (strong, nonatomic) IBOutlet UIView *noDevicesView;

@property (weak, nonatomic) IBOutlet UIBarButtonItem *addButton;

@property (weak, nonatomic) IBOutlet UILabel *footerLabel;

@end

@implementation DevicesTableViewController

- (void)viewDidLoad {
   [super viewDidLoad];

   //Load Devices on pull-down
   [self.refreshControl addTarget:self
                           action:@selector(loadDevicesList)
                 forControlEvents:UIControlEventValueChanged];
}

- (void)viewWillAppear:(BOOL)animated {
   [super viewWillAppear:animated];
   [self setAppearance];

   //Footer Label
   [_footerLabel setText:[NSString stringWithFormat:@"Voce pode adicionar no maximo %@
       dispositivos",
                        @(kSettingsMaxAuthorizedDevices)]];

   //Load Devices List
   [_addButton setEnabled:NO];
   [self loadDevicesList];
}

- (void) loadDevicesList {
   [self.refreshControl beginRefreshing];
   [General requestAllDevicesWithCompletion:^(BOOL success) {
      [self.refreshControl endRefreshing];
      if (success) {
         [self.tableView reloadData];
         //
         if (!SharedGeneral.allDevices.count) {
            [self.tableView setTableHeaderView:_noDevicesView];
            [_addButton setEnabled:YES];
         } else {
            [self.tableView setTableHeaderView:nil];
            if (SharedGeneral.allDevices.count >= kSettingsMaxAuthorizedDevices) {
```

```
                [_addButton setEnabled:NO];
            } else {
                [_addButton setEnabled:YES];
            }
        }
    }];
}

#pragma mark - Table View

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
    *)indexPath {
    static NSString *cellId = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellId];

    //Object
    PFObject *device = [SharedGeneral.allDevices objectAtIndex:indexPath.row];

    //Labels
    cell.textLabel.text = [device objectForKey:kParseColumnDeviceName];
    cell.detailTextLabel.text = [device objectForKey:kParseColumnDeviceModel];

    return cell;
}

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}

- (void)tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        PFObject *device = [SharedGeneral.allDevices objectAtIndex:indexPath.row];
        [device deleteInBackgroundWithBlock:^(BOOL succeeded, NSError * _Nullable error) {
            if (succeeded) {
                [SharedGeneral.allDevices removeObject:device];
                [tableView deleteRowsAtIndexPaths:@[indexPath]
                    withRowAnimation:UITableViewRowAnimationFade];
                [self loadDevicesList];
            }
        }];
    }
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
```

```
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
   return SharedGeneral.allDevices.count;
}

#pragma mark - Segue

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
   if ([segue.identifier isEqualToString:@"DeviceDetailsSegue"]) {
      DeviceDetailsTableViewController *viewController = segue.destinationViewController;
      NSIndexPath *selectedIndexPath = [self.tableView indexPathForSelectedRow];
      PFObject *device = [SharedGeneral.allDevices objectAtIndex:selectedIndexPath.row];
      [viewController setSelectedDevice:device];
   }
}

#pragma mark - Other

- (void) setAppearance {
   [_noDevicesView setBackgroundColor:[UIColor clearColor]];
   [_noDevicesView setFrame:CGRectMake(0, 0, self.view.frame.size.width, 200.0f)];
}

@end
```

Arquivo tipo ’interface’ da classe do singleton de uso geral - General.h:

```
//
// General.h
// SSV Mobile
//
// Created by Allan Alves on 3/25/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <Parse/Parse.h>

typedef void(^CompletionBlock)(BOOL success);

@interface General : NSObject

+ (instancetype) sharedGeneral;
- (instancetype) initPrivate;

@property (nonatomic) NSArray *allVehicleStatus;
@property (nonatomic) NSMutableArray *allDevices;
@property (nonatomic) NSArray *allEvents;
```

```
//Events
+ (void) requestAllEventsWithCompletion:(CompletionBlock)completion;
+ (PFObject*) eventWithNumber:(NSNumber*)number;


+ (void) requestAllVehicleStatusWithCompletion:(CompletionBlock)completion;
+ (void) requestAllDevicesWithCompletion:(CompletionBlock)completion;

@end
```

Arquivo tipo ’implementation’ da classe do singleton de uso geral - General.m:

```
//
// General.m
// SSV Mobile
//
// Created by Allan Alves on 3/25/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "General.h"

@implementation General

+ (instancetype)sharedGeneral {
   static General *shared = nil;
   static dispatch_once_t token;
   dispatch_once(&token, ^{
      shared = [[General alloc] initPrivate];
   });
   return shared;
}

- (instancetype)initPrivate {
   self = [super init];
   if (self) {
      _allDevices = [[NSMutableArray alloc] init];
   }
   return self;
}

#pragma mark - Parse - Event

+ (void)requestAllEventsWithCompletion:(CompletionBlock)completion {
   PFQuery *query = [PFQuery queryWithClassName:kParseTableEvent];
   [query findObjectsInBackgroundWithBlock:^(NSArray * _Nullable objects, NSError * _Nullable
       error) {
      if (error) {
```

```
            completion(NO);
        } else {
            SharedGeneral.allEvents = objects;
            completion(YES);
        }
    }];
}

+ (PFObject *)eventWithNumber:(NSNumber *)number {
    for (PFObject *event in SharedGeneral.allEvents) {
        NSNumber *eventNumber = [event objectForKey:kParseColumnEventNumber];
        if (eventNumber.integerValue == number.integerValue) {
            return event;
        }
    }
    return nil;
}

#pragma mark - Parse - Vehicle Status

+ (void)requestAllVehicleStatusWithCompletion:(CompletionBlock)completion {
    PFQuery *query = [PFQuery queryWithClassName:kParseTableVehicleStatus];
    [query orderByDescending:@"createdAt"];
    [query findObjectsInBackgroundWithBlock:^(NSArray * _Nullable objects, NSError * _Nullable
        error) {
        if (error) {
            completion(NO);
        } else {
            SharedGeneral.allVehicleStatus = objects;
            completion(YES);
        }
    }];
}

#pragma mark - Parse - Device

+ (void)requestAllDevicesWithCompletion:(CompletionBlock)completion {
    PFQuery *query = [PFQuery queryWithClassName:kParseTableDevice];
    [query orderByDescending:@"createdAt"];
    [query findObjectsInBackgroundWithBlock:^(NSArray * _Nullable objects, NSError * _Nullable
        error) {
        if (error) {
            completion(NO);
        } else {
            if (SharedGeneral.allDevices.count) {
                [SharedGeneral.allDevices removeAllObjects];
            }
            [SharedGeneral.allDevices addObjectsFromArray:objects];
            completion(YES);
```

```
      }
   }];
}

@end
```

Arquivo tipo ’plist’ de configurações gerais do projeto - Info.plist:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>pt-BR</string>
  <key>CFBundleExecutable</key>
  <string>$(EXECUTABLE_NAME)</string>
  <key>CFBundleIdentifier</key>
  <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>$(PRODUCT_NAME)</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleVersion</key>
  <string>1</string>
  <key>LSRequiresIPhoneOS</key>
  <true/>
  <key>UILaunchStoryboardName</key>
  <string>LaunchScreen</string>
  <key>UIMainStoryboardFile</key>
  <string>Main</string>
  <key>UIRequiredDeviceCapabilities</key>
  <array>
    <string>armv7</string>
  </array>
  <key>UISupportedInterfaceOrientations</key>
  <array>
    <string>UIInterfaceOrientationPortrait</string>
    <string>UIInterfaceOrientationLandscapeLeft</string>
    <string>UIInterfaceOrientationLandscapeRight</string>
  </array>
  <key>UIViewControllerBasedStatusBarAppearance</key>
  <false/>
```

```
  <key>NSAppTransportSecurity</key>
  <dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
  </dict>
</dict>
</plist>
```

Arquivo tipo ’pch’ chamado de ’PrefixHeader’ para código pré-compilado - PrefixHeader.pch:

```
//
// PrefixHeader.pch
// SSV Mobile
//
// Created by Allan Alves on 3/25/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#ifndef PrefixHeader_pch
#define PrefixHeader_pch

// Include any system framework and library headers here that should be included in all
    compilation units.
// You will also need to set the Prefix Header build setting of one or more of your targets
    to reference this file.

#endif /* PrefixHeader_pch */

#import "General.h"
#define SharedGeneral [General sharedGeneral]
#define SharedApplication [UIApplication sharedApplication]
#define StandardUserDefaults [NSUserDefaults standardUserDefaults]

#define kSettingsMaxAuthorizedDevices 2

#import "UIAlertController+Utils.h"

#define PARSE_APP_ID @"9Qv1SPg0W3jCz5bNlF7Ldny7w3yBYf1rIQlgvsr5"
#define PARSE_CLIENT_KEY @"FI2MASTgUaiUnMyBgkFrjc6NCARxWJhVK0DlO7hZ"

#define kParseTableVehicleStatus @"VehicleStatus"
#define kParseTableEvent @"Event"
#define kParseTableDevice @"Device"

//Vehicle Status Labels Descriptions
#define kVehicleStatusLabelIsLocked @"Veiculo trancado"
#define kVehicleStatusLabelEvent @"Evento no momento"
```

```
#define kVehicleStatusLabelMostRecent @"Registro mais recente"

// Parse Columns
#define kParseColumnLatitude @"latitude"
#define kParseColumnLongitude @"longitude"
#define kParseColumnIsLocked @"isLocked"
#define kParseColumnEvent @"event"

#define kParseColumnDeviceMacAddress @"macAddress"
#define kParseColumnDeviceName @"deviceName"
#define kParseColumnDeviceModel @"deviceModel"

#define kParseColumnEventNumber @"eventNumber"
#define kParseColumnEventDescription @"description"

/*! MODEL DETECTION */
#define IS_IPAD (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
#define IS_IPHONE (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone)
#define IS_RETINA ([[UIScreen mainScreen] scale] >= 2.0)

#define SCREEN_WIDTH ([[UIScreen mainScreen] bounds].size.width)
#define SCREEN_HEIGHT ([[UIScreen mainScreen] bounds].size.height)
#define SCREEN_MAX_LENGTH (MAX(SCREEN_WIDTH, SCREEN_HEIGHT))
#define SCREEN_MIN_LENGTH (MIN(SCREEN_WIDTH, SCREEN_HEIGHT))

#define IS_IPHONE_4_OR_LESS (IS_IPHONE && SCREEN_MAX_LENGTH < 568.0)
#define IS_IPHONE_5 (IS_IPHONE && SCREEN_MAX_LENGTH == 568.0)
#define IS_IPHONE_6 (IS_IPHONE && SCREEN_MAX_LENGTH == 667.0)
#define IS_IPHONE_6P (IS_IPHONE && SCREEN_MAX_LENGTH == 736.0)
```

Arquivo tipo ’interface’ da tela de histórico de situação do veículo - StatusHistoryViewController.h:

```
//
// StatusHistoryViewController.h
// SSV Mobile
//
// Created by Allan Alves on 3/26/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@interface StatusHistoryViewController : UIViewController

@end
```

Arquivo tipo ’implementation’ da tela de histórico de situação do veículo - StatusHistoryViewController.m:

```
//
// StatusHistoryViewController.m
// SSV Mobile
//
// Created by Allan Alves on 3/26/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "StatusHistoryViewController.h"

@interface StatusHistoryViewController ()

@property (weak, nonatomic) IBOutlet MKMapView *mapView;
@property (weak, nonatomic) IBOutlet UITableView *tableView;
@property (weak, nonatomic) IBOutlet UIView *noLocationView;

@property (weak, nonatomic) IBOutlet UILabel *isLockedLabel;
@property (weak, nonatomic) IBOutlet UILabel *eventLabel;

@end

@implementation StatusHistoryViewController

- (void)viewDidLoad {
   [super viewDidLoad];
}

- (void)viewDidAppear:(BOOL)animated {
   [super viewDidAppear:animated];

   //Select First Row of TableView
   NSIndexPath *firstRow = [NSIndexPath indexPathForRow:0 inSection:0];
   [self.tableView selectRowAtIndexPath:firstRow animated:YES
       scrollPosition:UITableViewScrollPositionBottom];
   [self.tableView.delegate tableView:self.tableView didSelectRowAtIndexPath:firstRow];
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
   return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
   return SharedGeneral.allVehicleStatus.count;
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
    *)indexPath {
   static NSString *cellId = @"Cell";
   UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellId
       forIndexPath:indexPath];

   PFObject *location = [SharedGeneral.allVehicleStatus objectAtIndex:indexPath.row];

   NSDateFormatter* dateFormatter = [[NSDateFormatter alloc] init];
   [dateFormatter setDateFormat:@"dd/MM/yyyy HH:mm"];
   NSString *dateDescription = [dateFormatter stringFromDate:location.createdAt];

   cell.detailTextLabel.text = dateDescription;

   NSNumber *eventNumber = [location objectForKey:kParseColumnEvent];
   cell.textLabel.text = [[General eventWithNumber:eventNumber]
                   objectForKey:kParseColumnEventDescription];

   return cell;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
   [self.mapView removeAnnotations:self.mapView.annotations];
   PFObject *vehicleStatusObject = [SharedGeneral.allVehicleStatus
       objectAtIndex:indexPath.row];

   //Get Info
   NSNumber *isLocked = [vehicleStatusObject objectForKey:kParseColumnIsLocked];
   NSNumber *eventNumber = [vehicleStatusObject objectForKey:kParseColumnEvent];
   //Is Locked Label
   NSString *isLockedLabelText = [NSString stringWithFormat:@"%@: ",
       kVehicleStatusLabelIsLocked];
   if (isLocked.boolValue) {
      isLockedLabelText = [isLockedLabelText stringByAppendingString:@"Sim"];
   } else {
      isLockedLabelText = [isLockedLabelText stringByAppendingString:@"ãNo"];
   }
   self.isLockedLabel.text = isLockedLabelText;
   //Event Label
   NSString *eventDescription = [[General eventWithNumber:eventNumber]
                         objectForKey:kParseColumnEventDescription]; //Event Descr.

   NSString *eventLabelText = [NSString stringWithFormat:@"%@: %@",
                       kVehicleStatusLabelEvent, eventDescription];

   self.eventLabel.text = eventLabelText; //Set Label Text

   //Get Location
```

```
    NSString *lat = [vehicleStatusObject objectForKey:kParseColumnLatitude];
    NSString *lng = [vehicleStatusObject objectForKey:kParseColumnLongitude];

    if (lat.length && lng.length) {
        CLLocation *location = [[CLLocation alloc] initWithLatitude:lat.doubleValue
                                                          longitude:lng.doubleValue];
        [self addAnnotationForLocation:location];
        [self zoomToCoordinate:location.coordinate];
        [self.mapView setHidden:NO];
    } else {
        [self.noLocationView setHidden:NO];
        [self.mapView setHidden:YES];
    }
}

#pragma mark - Map

- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id
     <MKAnnotation>)annotation {
    MKPinAnnotationView *annotationView;
    if(!annotationView) {
        annotationView = (MKPinAnnotationView*)[[MKPinAnnotationView alloc]
            initWithAnnotation:annotation reuseIdentifier:@"String"];
        annotationView.animatesDrop = YES;
        if ([annotationView respondsToSelector:@selector(setPinTintColor:)]) {
            annotationView.pinTintColor = [UINavigationBar appearance].barTintColor;
        }
        annotationView.rightCalloutAccessoryView = [UIButton
            buttonWithType:UIButtonTypeDetailDisclosure];
        [annotationView.rightCalloutAccessoryView setBackgroundColor:[UIColor clearColor]];
        [annotationView.rightCalloutAccessoryView setTintColor:[[UIView appearance] tintColor]];
    }
    annotationView.enabled = YES;
    annotationView.canShowCallout = YES;

    return annotationView;
}

#pragma mark - Location

- (void) addAnnotationForLocation:(CLLocation*)location {
    MKPointAnnotation *point = [[MKPointAnnotation alloc] init];
    point.coordinate = location.coordinate;
    [self.mapView addAnnotation:point];
}

- (void) zoomToCoordinate:(CLLocationCoordinate2D)coordinate {
    CLLocationCoordinate2D noLocation = coordinate;
    MKCoordinateRegion viewRegion = MKCoordinateRegionMakeWithDistance(noLocation, 500, 500);
```

```
   MKCoordinateRegion adjustedRegion = [self.mapView regionThatFits:viewRegion];
   [self.mapView setRegion:adjustedRegion animated:NO];
}

@end
```

Arquivo tipo ’interface’ da tela de situação do veículo - StatusViewController.h:

```
//
// StatusViewController.h
// SSV Mobile
//
// Created by Allan Alves on 3/26/16.
// Copyright (c) 2016 AA. All rights reserved.
//


#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import <Parse/Parse.h>

@interface StatusViewController : UIViewController <MKMapViewDelegate>

@end
```

Arquivo tipo ’implementation’ da tela de situação do veículo - StatusViewController.m:

```
//
// StatusViewController.m
// SSV Mobile
//
// Created by Allan Alves on 3/26/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "StatusViewController.h"

@interface StatusViewController ()

@property (weak, nonatomic) IBOutlet UIActivityIndicatorView *loading;
@property (weak, nonatomic) IBOutlet MKMapView *mapView;
@property (weak, nonatomic) IBOutlet UIView *noLocationView;
@property (weak, nonatomic) IBOutlet UIBarButtonItem *refreshButton;

@property (weak, nonatomic) IBOutlet UILabel *mostRecentLabel;
@property (weak, nonatomic) IBOutlet UILabel *isLockedLabel;
@property (weak, nonatomic) IBOutlet UILabel *eventLabel;
```

```
@end

@implementation StatusViewController {
   CLLocation *lastLocation; //Last location found on Cloud
   PFObject *lastVehicleStatusObject; //Last Parse Vehicle Status Object
}

- (void)viewDidLoad {
   [super viewDidLoad];
   [self.mapView setHidden:YES];
   [self setAppearance];
   [self addObservers];

   if (SharedGeneral.allEvents) {
      [self requestVehicleStatuses];
   }
}

- (void) requestVehicleStatuses {
   [General requestAllVehicleStatusWithCompletion:^(BOOL success) {
      if (success) {
         //Enable Refresh Button
         [_refreshButton setEnabled:YES];

         //Get Most Recent Object
         lastVehicleStatusObject = SharedGeneral.allVehicleStatus.firstObject;

         //Get Info
         NSNumber *isLocked = [lastVehicleStatusObject objectForKey:kParseColumnIsLocked];
         NSNumber *eventNumber = [lastVehicleStatusObject objectForKey:kParseColumnEvent];

         //Is Locked Label
         NSString *isLockedLabelText = [NSString stringWithFormat:@"%@: ",
             kVehicleStatusLabelIsLocked];
         if (isLocked.boolValue) {
            isLockedLabelText = [isLockedLabelText stringByAppendingString:@"Sim"];
         } else {
            isLockedLabelText = [isLockedLabelText stringByAppendingString:@"ãNo"];
         }
         self.isLockedLabel.text = isLockedLabelText;

         //Event Label
         //Event Label
         NSString *eventDescription = [[General eventWithNumber:eventNumber]
                                       objectForKey:kParseColumnEventDescription]; //Event Descr.

         NSString *eventLabelText = [NSString stringWithFormat:@"%@: %@",
                                     kVehicleStatusLabelEvent, eventDescription];
```

```
        self.eventLabel.text = eventLabelText; //Set Label Text

        //Most Recent Label
        NSDateFormatter* dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat:@"dd/MM/yyyy HH:mm"];
        NSString *dateDescription = [dateFormatter
            stringFromDate:lastVehicleStatusObject.createdAt];

        self.mostRecentLabel.text = [NSString stringWithFormat:@"%@ (%@)",
                              kVehicleStatusLabelMostRecent,
                              dateDescription];

        //Get Last Location Registered
        NSString *latitude = [lastVehicleStatusObject objectForKey:kParseColumnLatitude];
        NSString *longitude = [lastVehicleStatusObject objectForKey:kParseColumnLongitude];

        //Create CLLocation
        lastLocation = [[CLLocation alloc]
                    initWithLatitude:latitude.doubleValue
                    longitude:longitude.doubleValue];

        //Fade In Map or No Location Warning
        if (latitude.length && longitude.length) {
           [self fadeInMap];
        } else {
           [self fadeInNoLocationView];
        }
     } else {
        [UIAlertController simpleAlertInViewController:self
                                    withTitle:@"çõLocalizaes"
                                   andMessage:@"Ocorreu um erro ao buscar as çõlocalizaes
                                       registradas."];
     }
   }];
}

- (void) fadeInNoLocationView {
   self.noLocationView.alpha = 0;
   [self.noLocationView setHidden:NO];
   [UIView animateWithDuration:1.0f animations:^{
      self.noLocationView.alpha = 1.0f;
      self.mapView.alpha = 0;
      self.loading.alpha = 0;
   } completion:^(BOOL finished) {
      [self.mapView setHidden:YES];
      [self.loading stopAnimating];
      [self.navigationItem.leftBarButtonItem setEnabled:YES];
      [self.navigationItem.rightBarButtonItem setEnabled:YES];
```

```
    }];
}

- (void) fadeInMap {
    self.mapView.alpha = 0;
    self.mapView.hidden = NO;
    [UIView animateWithDuration:1.0f animations:^{
        self.mapView.alpha = 1.0f;
        self.noLocationView.alpha = 0;
        self.loading.alpha = 0;
    } completion:^(BOOL finished) {
        [self.noLocationView setHidden:YES];
        [self addAnnotationForLocation:lastLocation];
        [self zoomToCoordinate:lastLocation.coordinate];
        [self.loading stopAnimating];
        [self.navigationItem.leftBarButtonItem setEnabled:YES];
        [self.navigationItem.rightBarButtonItem setEnabled:YES];
    }];
}

#pragma mark - Actions

- (IBAction)refreshLocations:(id)sender {
    //Disable Refresh Button
    [_refreshButton setEnabled:NO];

    self.loading.alpha = 0;
    [self.loading setHidden:NO];
    [UIView animateWithDuration:0.3f animations:^{
        self.mapView.alpha = 0;
        [self.loading startAnimating];
        self.loading.alpha = 1.0f;
    } completion:^(BOOL finished) {
        [self.mapView removeAnnotations:self.mapView.annotations];
        [self.mapView setHidden:YES];
        [self requestVehicleStatuses];
    }];
}

#pragma mark - Map

- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id
    <MKAnnotation>)annotation {
    MKPinAnnotationView *annotationView;
    if(!annotationView) {
        annotationView = (MKPinAnnotationView*)[[MKPinAnnotationView alloc]
            initWithAnnotation:annotation reuseIdentifier:@"String"];
        annotationView.animatesDrop = YES;
        if ([annotationView respondsToSelector:@selector(setPinTintColor:)]) {
```

```
        annotationView.pinTintColor = [UIColor redColor];
      }
   }
   annotationView.enabled = YES;
   annotationView.canShowCallout = YES;

   return annotationView;
}

#pragma mark - Location

- (void) addAnnotationForLocation:(CLLocation*)location {
   MKPointAnnotation *point = [[MKPointAnnotation alloc] init];
   point.coordinate = location.coordinate;
   point.title = @"Registro mais recente";

   NSDateFormatter* dateFormatter = [[NSDateFormatter alloc] init];
   [dateFormatter setDateFormat:@"dd/MM/yyyy HH:mm"];
   NSString *dateDescription = [dateFormatter
       stringFromDate:lastVehicleStatusObject.createdAt];

   NSString *subtitle = dateDescription;
   point.subtitle = subtitle;

   [self.mapView addAnnotation:point];
}

- (void) zoomToCoordinate:(CLLocationCoordinate2D)coordinate {
   CLLocationCoordinate2D noLocation = coordinate;
   MKCoordinateRegion viewRegion = MKCoordinateRegionMakeWithDistance(noLocation, 500, 500);
   MKCoordinateRegion adjustedRegion = [self.mapView regionThatFits:viewRegion];
   [self.mapView setRegion:adjustedRegion animated:NO];
}

#pragma mark - Appearance

- (void) setAppearance {
   [self.loading setColor:[[UINavigationBar appearance] barTintColor]];
}

#pragma mark - Other

- (void) addObservers {
   //Request All Vehicle Status when fetching events is done
   [[NSNotificationCenter defaultCenter] addObserver:self
                                    selector:@selector(requestVehicleStatuses)
                                       name:@"AllEventsLoaded"
                                     object:nil];
}
```

```
@end
```

Arquivo tipo ’interface’ de uma ’category’ da classe ’UIAlertController’ para simplificar implementação de alertas - UIAlertController+Utils.h:

```
//
// UIAlertController+Utils.h
// SSV Mobile
//
// Created by Allan Alves on 3/25/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface UIAlertController (Utils)

+ (void) simpleAlertInViewController:(UIViewController*)viewController
                     withTitle:(NSString*)title
                    andMessage:(NSString*)message;

@end
```

Arquivo tipo ’implementation’ de uma ’category’ da classe ’UIAlertController’ para simplificar implementação de alertas - UIAlertController+Utils.m:

```
//
// UIAlertController+Utils.m
// SSV Mobile
//
// Created by Allan Alves on 3/25/16.
// Copyright (c) 2016 AA. All rights reserved.
//

#import "UIAlertController+Utils.h"

@implementation UIAlertController (Utils)

+ (void)simpleAlertInViewController:(UIViewController *)viewController
                    withTitle:(NSString *)title
                   andMessage:(NSString *)message {
   UIAlertController *alert = [UIAlertController alertControllerWithTitle:title
                                                                   message:message
                                                            preferredStyle:UIAlertControllerStyleAlert];
   UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"OK"
       style:UIAlertActionStyleDefault handler:nil];
```

```
    [alert addAction:okAction];
    [viewController presentViewController:alert animated:YES completion:nil];
}

@end
```