UNIVERSIDADE REGIONAL INTEGRADA DO ALTO URUGUAI E DAS MISSÕES - URI ERECHIM
DEPARTAMENTO DE ENGENHARIAS E CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ELIAS SARTORI

BUS FINDER - APLICATIVO MÓVEL PARA RASTREAMENTO DE VEÍCULOS DE TRANSPORTE ESCOLAR

ERECHIM - RS
2019

ELIAS SARTORI

BUS FINDER - APLICATIVO MÓVEL PARA RASTREAMENTO DE VEÍCULOS DE TRANSPORTE ESCOLAR

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel, Departamento de Engenharias e Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões - URI Erechim.

Orientador: Prof. Dr. Guilherme Afonso Madalozzo

ERECHIM - RS
2019

AGRADECIMENTOS

Agradeço aos meus pais por todo o esforço investido na minha educação, sem eles com certeza não teria chegado até aqui.

Sou grato pela confiança depositada na minha proposta de projeto pelo meu professor Dr. Guilherme Afonso Madalozzo, orientador do meu trabalho. Obrigado por me manter motivado durante todo o processo.

Aos meus colegas do curso de Ciência da Computação, pelas trocas de ideias e ajuda mútua. Juntos, conseguimos avançar e ultrapassar todos os obstáculos.

Por último, quero agradecer também à Universidade Regional Integrada do Alto Uruguai e das Missões - URI Erechim e todo o seu corpo docente.

"Nunca se compare com ninguém neste mundo. Caso o faça, entenda que você estará insultando a si mesmo."

(Bill Gates)

RESUMO

Rastreamento veicular através de geolocalização já é algo muito difundido em vários segmentos como: monitoramento de cargas, frotas de táxi, carros fortes e em diversas outras situações. Visto isso, é possível perceber que a área de transportes escolares ainda foi pouco explorada com esta finalidade, com foco na cidade de Erechim, no estado do Rio Grande do Sul, no Brasil. Levando em consideração estes fatores e, aliado a isso o crescimento exponencial dos sistemas embarcados e dos aplicativos móveis, o presente trabalho desenvolve um aplicativo móvel para realizar o rastreamento de veículos que realizam transporte escolar. Este aplicativo, tem como principais funcionalidades: rastrear em tempo real a posição do veículo em questão e a distância de trajeto a ser percorrido em relação ao usuário. A ferramenta utilizada para o desenvolvimento do aplicativo foi *JavaScript* com *React Native*, recebendo dados de uma API em *Ruby*, com *Ruby on Rails*. Para obter os dados de geolocalização, foi utilizado uma placa arduino, com módulo GSM/GPRS.

Palavras-chave: Rastreamento Veicular, Arduíno, Aplicação Móvel.

ABSTRACT

Vehicle tracking through geolocation is already very widespread in various segments such as cargo monitoring, taxi fleets, security vans and various other situations. Considering this, it is possible to notice that the area of school transport was still very little explored for this purpose, with focus at Erechim, state of Rio Grande do Sul, Brazil. Taking into account these factors, and allied to the exponential growth of embedded systems and mobile applications, the present work develops a mobile application to track vehicles that carry school transportation. This application has as main features, track in real time the position of the vehicle in question and the distance to be traveled in relation to the user. The tool used for application development was JavaScript with React Native, receiving data from a Ruby API with Ruby on Rails. To obtain the geolocation data, an arduino board with GSM/GPRS module was used.

Keywords: Vehicle tracking, Arduino, Mobile Application.

LISTA DE ILUSTRAÇÕES

Figura 1 – Padrões utilizados antes do GSM. 15
Figura 2 – Resumo do desenvolvimento do padrão GSM. 16
Figura 3 – Arquitetura de rede GSM 17
Figura 4 – Arquitetura GPRS. 19
Figura 5 – Exemplo do Arduino Uno. 22
Figura 6 – Exemplo do Arduino Mega. 23
Figura 7 – Exemplo do Arduino Nano 23
Figura 8 – Exemplo do Arduino Mini. 24
Figura 9 – Exemplo de código arduino. 25
Figura 10 – Estrutura GPRS/GSM *Shield* SIM900. 30
Figura 11 – Estrutura traseira GPRS/GSM *Shield* SIM900. 30
Figura 12 – Exemplo *Google Maps* 35
Figura 13 – Modulo GSM GPRS Shield EFCom SIM900 com Arduino Uno. 36
Figura 14 – Função *setup()* 37
Figura 15 – Função *loop()* 38
Figura 16 – Estrutura do projeto *Ruby on Rails* 39
Figura 17 – Modelo entidade-relacionamento 40
Figura 18 – Função que retorna dados de coordenadas da API 40
Figura 19 – Função que insere dados de coordenadas da API 41
Figura 20 – Função que cria novo usuário na API 41
Figura 21 – Modelo que cria chave de acesso ao usuário 42
Figura 22 – Autenticação na API 42
Figura 23 – Controlador de sessão na API 43
Figura 24 – Diagrama de atividade 44
Figura 25 – Criação de sessão na API 45
Figura 26 – Tela de entrada do aplicativo. 46
Figura 27 – Função que cria novo usuário. 47
Figura 28 – Tela de criação de conta. 48
Figura 29 – Tela inicial do aplicativo. 49
Figura 30 – Função que busca dados sobre veículos. 49
Figura 31 – Tela que mostra mapa de rastreamento do veículo. 50
Figura 32 – Função que busca as coordenadas do ônibus. 51
Figura 33 – Função que atualiza posição do ônibus. 52
Figura 34 – Função que obtém posição do usuário e que calcula distância. 52

LISTA DE TABELAS

Tabela 1 – Modificações GSM para suportar GPRS 18

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicações, do inglês *Application Programming Interface*
AUC	Centro de autenticação, do inglês *Authentication Center*
BSS	Subsistema de Estação Base, do inglês *Base Station Subsystem*
BTS	Estação Radio Base, do inglês *Base Transceiver Station*
CEPT	Conferência Europeia de Administração de Correios e Telecomunicações, do francês *Conférence Européenne des Postes et Télécommunications*
EDGE	*Enhanced Data rates for GSM Evolution*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
EIR	Registro de identidade de equipamento, do inglês *Equipment Identity Register*
FTDI	*Future Technology Devices International*
GPRS	Serviços Gerais de Pacote por Rádio, do inglês *General Packet Radio Services*
GPS	Sistema de posicionamento global, do inglês *Global Position System*
GSM	Sistema Global para Comunicações Móveis, do inglês *Global System for Mobile*
HLR	Registro de localização local, do inglês *Home Location Register*
HTTPS	Protocolo de transferência de hipertexto seguro, do inglês *Hyper Text Transfer Protocol Secure*
IMEI	Identidade internacional de equipamento móvel, do inglês *International Mobile Equipment Identity*
IP	Protocolo de Internet, do inglês *Internet Protocol*
JSON	*JavaScript Object Notation*
JSX	*JavaScript eXtension*
KB	Kilobyte

LCD	Display de cristal líquido, do inglês *liquid crystal display*
MHz	Megahertz
MS	Estação móvel, do inglês *Mobile Station*
MSC	*Mobile Switching System*
NSS	Subsistema de rede e comutação, do inglês *Network and Switching Subsystem*
OMS	Sistema de Operações e Manutenção, do inglês *Operations and Maintenance System*
PCU	*Packet Control Unit*
PWM	Modulação por largura de pulso, do inglês *Pulse Width Modulation*
REST	Transferência de Estado Representacional, do inglês *Representational State Transfer*
SIM	Módulo de identidade do assinante, do inglês *Subscriber Identity Module*
SQL	Linguagem de Consulta Estruturada, do inglês *Structured Query Language*
SRAM	Memória estática de acesso aleatório, do inglês *Static Random-access Memory*
UMTS	*Universal Mobile Telecommunication System*
VLR	Registro de localização de visitante, do inglês *Visitor Location Register*

SUMÁRIO

1 INTRODUÇÃO 13

2 TECNOLOGIAS 14
2.1 Redes GSM/GPRS 14
2.1.1 Histórico 14
2.1.2 Desenvolvimento do padrão GSM 15
2.1.3 Arquitetura de rede GSM 16
2.1.4 Tecnologia GPRS 18
2.1.5 Arquitetura GPRS 18
2.2 Rastreadores 19
2.2.1 Rastreamento veicular 19
2.2.1.1 Tipos de Rastreadores Veiculares 20
2.3 Arduino 20
2.3.1 O que é um Arduino? 21
2.3.2 Origem do Arduino 21
2.3.3 Exemplos de modelos 21
2.3.3.1 Arduino Uno 21
2.3.3.2 Arduino Mega 22
2.3.3.3 Arduino Nano 23
2.3.3.4 Arduino Mini 23
2.3.4 Arduino IDE 24
2.3.5 Aplicabilidade 25
2.4 Aplicativos Móveis 25
2.4.1 Conceitos básicos sobre aplicativos móveis 25
2.4.2 Dados estatísticos sobre aplicativos móveis 26
2.4.3 Desenvolvimento de Aplicativos Móveis 26
2.4.3.1 Aplicativos nativos 27
2.4.3.2 *Web Apps* 27
2.4.3.3 Aplicativos híbridos 28

3 DESENVOLVIMENTO 29
3.1 Ferramentas Utilizadas 29
3.1.1 GPRS/GSM *Shield* SIM900 29
3.1.1.1 Estrutura da placa 29
3.1.2 Linguagem de programação *Ruby* 31
3.1.2.1 Tipo de dados 31

3.1.3 *Ruby On Rails* . . . 31
3.1.3.1 História . . . 32
3.1.3.2 Arquitetura . . . 32
3.1.3.3 Convenções do *Ruby On Rails* . . . 32
3.1.3.4 Principais componentes do *Ruby On Rails* . . . 32
3.1.4 *PostgreSQL* . . . 33
3.1.5 *React Native* . . . 33
3.1.6 *Google Maps API* . . . 34
3.2 Montagem e configuração do módulo *GSM GPRS Shield EFCom SIM900* 35
3.3 Desenvolvimento da API . . . 38
3.3.1 Estrutura e configurações iniciais do projeto . . . 38
3.3.2 Modelo entidade-relacionamento . . . 39
3.3.3 Descrição de implementações do projeto . . . 40
3.4 Desenvolvimento da Aplicação Móvel . . . 43
3.4.1 Modelagem UML . . . 43
3.4.2 Estrutura inicial do projeto . . . 44
3.4.3 Descrição de telas e implementações . . . 44

4 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS . . . 53

REFERÊNCIAS . . . 54

1 INTRODUÇÃO

O rastreamento veicular é um segmento que possui uma plena consolidação. Isto deve-se ao fato de que muitas empresas procuram usar geolocalização para monitorar a localização da sua frota, com isso aumentando consideravelmente a sua segurança. Porém, este serviço não está ligado somente a esse segmento. O rastreamento veicular também é utilizado para transportes coletivos, motoristas de aplicativos, frotas de táxi e até em carros particulares.

Assim como o rastreamento, o ramo de aplicativos móveis também já está difundido em praticamente todas as áreas da atual sociedade. A mobilidade e a rapidez com que as informações são acessadas através de aplicações móveis geram uma otimização do tempo e consequentemente melhoram a rotina de todos.

Com isso, o presente trabalho foi capaz de desenvolver uma solução que necessitava de tecnologias diferentes e assim gerando conhecimento sobre as mesmas. Foi desenvolvido um aplicativo móvel com a ferramenta *React Native*, consumindo dados de uma API escrita em *Ruby*, com o *framework Ruby On Rails* e este recebendo informações de um sistema embarcado projetado com um módulo GSM/GPRS SIM900 acoplado ao Arduino Uno R3.

No decorrer do trabalho, o segundo Capítulo aborda as tecnologias utilizadas. Iniciando com as redes GSM/GPRS, explana o seu histórico, desenvolvimento do padrão GSM, arquitetura de rede GSM e a tecnologia GPRS com sua arquitetura. Depois disso, aborda os rastreadores veiculares, seus conceitos básicos e tipos. Ainda no mesmo capítulo é definido o que é o arduino, suas origens, seus exemplos de modelos, ferramenta Arduino IDE e suas aplicabilidades. E por fim, apresenta-se as características dos aplicativos móveis, dados estatísticos e técnicas de desenvolvimento.

Por último, o terceiro Capítulo, traz toda a descrição do ferramental utilizado no desenvolvimento deste trabalho, tais como: módulo GRPS/GSM SIM900, linguagem de programação *Ruby*, o *framework Ruby On Rails*, banco de dados *PostgreSQL* e *Google Maps* API. Após isso, é demostrado todo o desenvolvimento do projeto, descrevendo a montagem e configuração do módulo *GSM GPRS Shield EFCom SIM900*, desenvolvimento da API e construção do aplicativo móvel.

2 TECNOLOGIAS

Este capítulo aborda as tecnologias utilizadas no decorrer do trabalho.

2.1 Redes GSM/GPRS

Esta seção tem como objetivo apresentar o histórico, padrão, arquitetura e modo de utilização das tecnologias GSM/GPRS, utilizadas para o desenvolvimento deste trabalho.

2.1.1 Histórico

O primeiro celular comercial foi o NMT 450 (*Nordic Mobile Telephone*) e surgiu em 1981, na Suécia e posteriormente disponibilizado para os demais países nórdicos. No entanto, no início dos anos 1980 era tratado apenas como um serviço auxiliar. Na época, não havia nenhum prognóstico do crescimento exponencial que ocorreu nos próximos anos. (SILVA; ROCHA; FREIRE, 2006)

Após o NMT 450 ser lançado, a tecnologia foi se espalhando pelo resto da Europa, mas de uma forma muito desordenada. Muitos países desenvolveram o seu sistema, mas não havia nenhum tipo de padronização. Com isso, cada país adotou suas próprias regras técnicas, tais como: protocolos de comunicação, potência de transmissão, largura de banda, entre outros. Desta maneira gerando uma divergência de padrões entre os países europeus. Visto isso, não era possível utilizar telefones celulares em um país com outros padrões técnicos, o que acabou se tornando um problema crítico após o aumento de usuários. A Figura 1 mostra os padrões detalhados em cada país.(PIROTTI; ZUCCOLOTTO, 2009)

Figura 1 – Padrões utilizados antes do GSM.

Sistema	Nordic Mobile Telephone	Total Access Communication System	Nordic Mobile Telephone	Radicom 2000
Início de operação	1981	1985	1986	1985
Frequência de transmissão do terminal móvel (MHz)	450	890	890	VHF, UHF e 900
Frequência de transmissão da estação rádio-base (MHz)	460	935	935	VHF, UHF e 900
Largura de Banda (MHz)	4.5	2 x 7,5	2,5	3,2
Largura dos canais (KHz)	25	25	25	12,5
Número de canais	180	2 x 300	1000	256
Países que utilizam	Suécia,Noruega, Dinamarca, Finlândia, Islandia,Áustria, Bélgica, Espanha, Luxemburgo,Holanda	Reino Unido, Áustria, Espanha, Irlanda, Itália	Suécia, Noruega, Dinamarca, Filândia,Holanda, Suíça	França

Fonte: (SILVA; ROCHA; FREIRE, 2006)

2.1.2 Desenvolvimento do padrão GSM

Com a existência de diferentes padrões nos países europeus, a integração entre os sistemas dos telefones celulares era iminente. Com isso, a criação de um novo padrão começou a ser idealizada, reunindo poderes públicos envolvendo diferentes países e operadoras de serviço móvel que utilizariam este novo padrão. (SILVA; ROCHA; FREIRE, 2006)

Os primeiros passos para isso ocorreram em 1982, em Estocolmo, onde se encontraram representantes de 11 países europeus, afim de criar um grupo, dentro do CEPT, para desenvolver a padronização do novo sistema chamado *Group Special Mobile* (GSM). A partir disso, começaram as primeiras discussões, criando três grupos de trabalho: o primeiro para definição dos serviços a serem oferecidos, segundo para especificação da transmissão de rádio e o terceiro para arquitetura de rede, protocolos de sinalização e interfaces entre diversas partes do sistema. Cada um desses grupos atuava de forma autônoma dentro de sua área. (PIROTTI; ZUCCOLOTTO, 2009)

Após os primeiros anos de pesquisa, em 1985 foi criado uma lista de normas com aproximadamente 100 tópicos segmentados em 12 áreas. A pesquisa continuou e, em 1991, a versão final da lista de normas do novo padrão foi criada, totalizando 130 tópicos e mais de 5000 páginas. Mais tarde o grupo alterou a sua nomenclatura, passando a se chamar *Global System for Mobile Communicatios* e preservando a sigla GSM. Segue figura 2, que detalha o resumo do desenvolvimento do padrão GSM. (SILVA; ROCHA; FREIRE, 2006)

Figura 2 – Resumo do desenvolvimento do padrão GSM.

ANO	EVENTO
1982-1985	Especificação por parte da CEPT de um padrão de telecomunicações digital europeu na faixa de 900MHz, conhecido como GSM.
1986	Escolha do TDMA e FDMA como tecnologias de transmissão.
1987	Operadoras de 12 países assinam um Memorando de Comprometimento, para implantação do GSM até 1991.
1988	CEPT define especificações do GSM para implementação em fases.
1989	ETSI assumiu a responsabilidade pela especificação do GSM.
1990	Fabricantes começam a desenvolver equipamentos de rede.
1991	Lançado padrão GSM 1800.
1992	Lançada primeira fase comercial de redes GSM. Definido primeiro acordo de roaming internacional entre a Telecom da Finlândia e a Vodafone da Inglaterra.
1993	GSM passa a ter 70 países signatários. Lançado sistema DCS 1800 na Inglaterra.
1993	Número de usuários chega a 3 milhões.
1995	Desenvolvida nos EUA, a especificação para Serviços de Comunicação Pessoais (PCS), versão do GSM para a faixa de 1900MHz.
1998	Rede GSM com um total de 253 membros em mais de 100 países com 70 milhões de usuários no mundo.
2000	Implantação no Brasil do SMP (Serviço Móvel Pessoal), na faixa de 1800MHz.

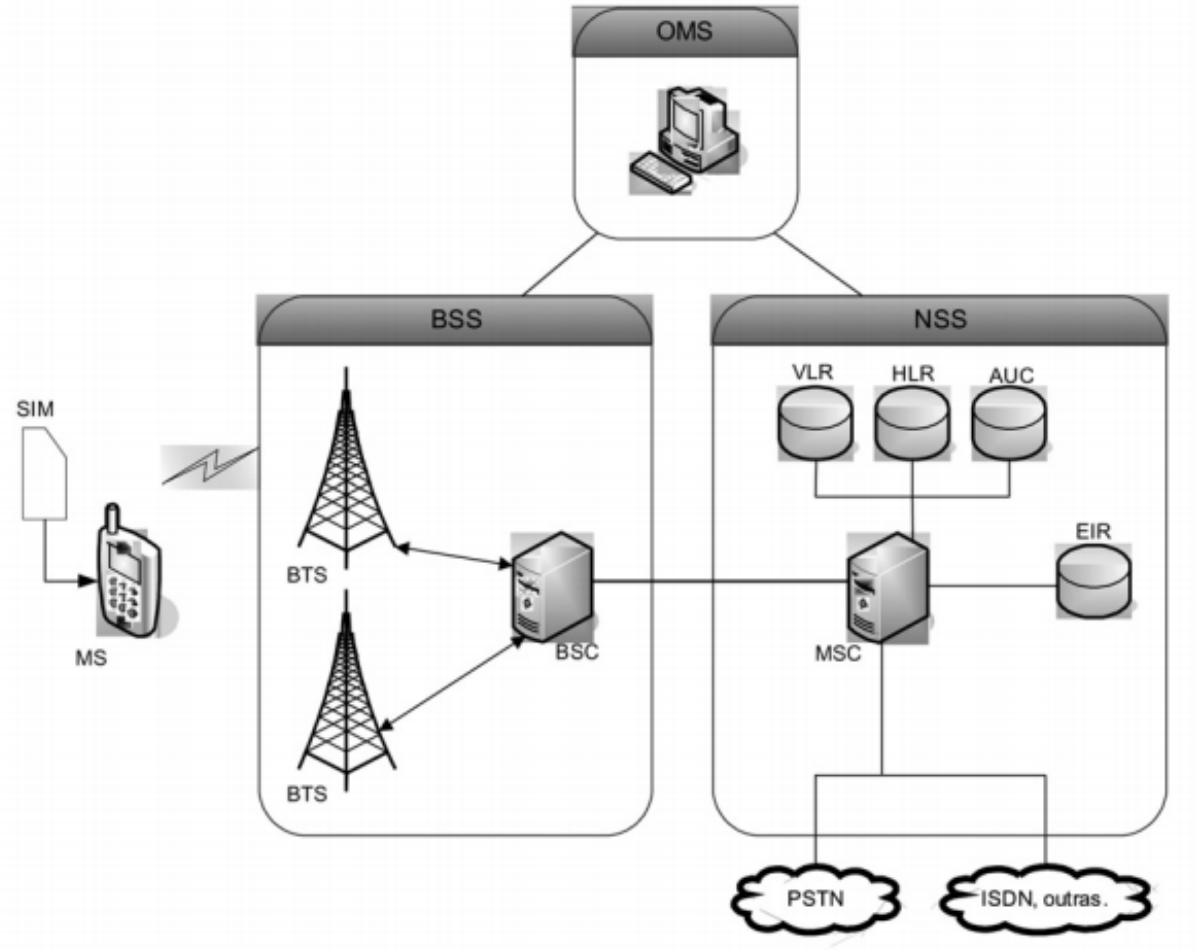
Fonte: (PIROTTI; ZUCCOLOTTO, 2009)

2.1.3 Arquitetura de rede GSM

A arquitetura de rede é dividida em três partes: BSS (*Base Station Subsystem*), NSS (*Network and Switching Subsystem*) e OMS (*Operations and Maintenance System*). O BSS é a estrutura aérea do sistema, que fornece acesso dos usuários aos rádios das estações base para controle e monitoramento dos mesmos. O NSS é onde ocorre o processamento de chamadas, tais como: base de dados de usuários, autenticação, conexão entre outros, ou seja, é a parte de gerenciamento do sistema. O OSS é a parte de suporte do sistema, responsável por monitorar e tratar erros vindos da rede. A Figura 3 ilustra a arquitetura de rede GSM. (PIROTTI; ZUCCOLOTTO, 2009)

A MS (*Mobile Station*) é um consumidor final da rede GSM, ou seja, os próprios telefones celulares ou qualquer outro dispositivo com capacidade de se conectar com a rede. O BTS (*Base Transceiver Station*) possui a função de fazer a comunicação via interface aérea, popularmente conhecida como antena. Uma MS busca sempre encontrar uma BTS com o melhor sinal disponível e com maior canal de frequência, com isso a potência exigida é menor. (PIROTTI; ZUCCOLOTTO, 2009)

Figura 3 – Arquitetura de rede GSM



Fonte: (PIROTTI; ZUCCOLOTTO, 2009)

O cartão SIM (*Subscriber Identity Module*) fornece informações de identificação da MS durante a conexão com a rede GSM. Porém este identificador não fica armazenado no próprio cartão SIM, mas na operadora de telefonia celular da rede. (PIROTTI; ZUCCOLOTTO, 2009)

A MSC (*Mobile Switching System*) é responsável por gerenciar a troca de chamadas e mensagem entre a MS e a rede GSM. No entanto, o HLR (*Home Location Register*) realiza o gerenciamento dos dados dos assinantes da rede, tais como: autenticação, localização, número do assinante e serviços disponíveis para o usuário. O AUC (*Authentication Center*) faz a autenticação e criptografia da rede, trabalhando junto com o HLR. O VLR (*Visitor Location Register*) mantém de forma temporária informações sobre os assinantes que estão conectados na rede e, na maioria dos casos, faz parte do MSC ou se encontra num equipamento a parte, isto em caso de grande tamanho da rede ou número de usuários elevados. Por fim, o EIR (*Equipment Identity Register*) é uma base de dados responsável por armazenar a Identidade Internacional do Equipamento Móvel (IMEI) das MSs conectadas. (SILVA; ROCHA; FREIRE, 2006)

2.1.4 Tecnologia GPRS

Como a rede GSM foi desenvolvida inicialmente para apenas prover serviços de chamadas de voz, não foi grande o tempo para essa demanda aumentar. Dessa forma, os assinantes da rede passaram a possuir a necessidade de trafegar dados mais complexos, tais como enviar mensagens de texto, imagens, acessar páginas na internet entre outros. (SVERZUT, 2015)

Para suprir a necessidade dessa evolução novos serviços foram oferecidos nas redes GSM. Com isso, já se encontram disponíveis serviços tais como 2,5G GPRS , 2,75G EDGE e o sistema de terceira geração 3G UMTS. A tecnologia GPRS utiliza a rede GSM como base, adicionando a ela uma estrutura capaz de acessar a internet e, com isso, foi desenvolvendo a terceira geração do padrão de telefonia móvel (3G). (PIROTTI; ZUCCOLOTTO, 2009)

2.1.5 Arquitetura GPRS

Para viabilizar a implementação do serviço GPRS foram necessários adicionar elementos e nova interface na arquitetura de rede GSM, que anteriormente era da segunda geração (2G), com o intuito de prover o transporte de dados por pacote IP e com isso migrando para a terceira geração.(SVERZUT, 2015)

A Tabela 1 aponta as mudanças necessárias para a rede GSM suportar o serviço GPRS.

Tabela 1 – Modificações GSM para suportar GPRS

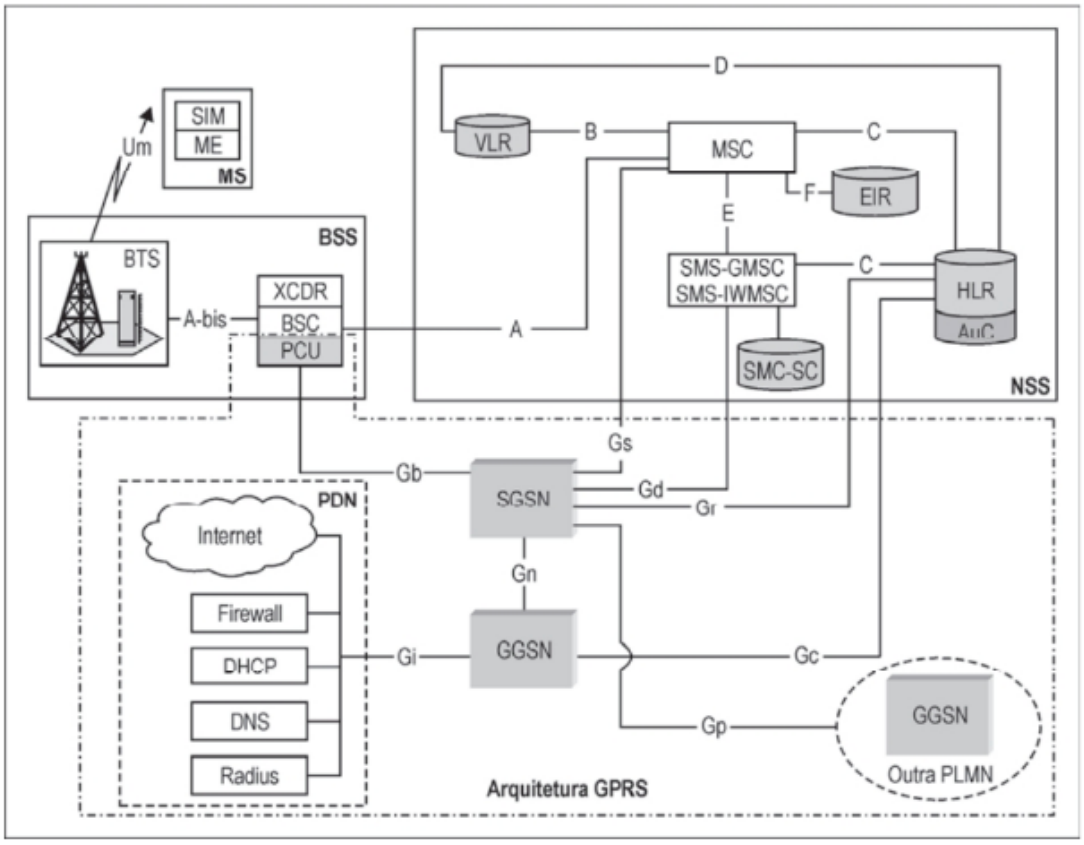
Elemento de rede	**Modificações**
MS	A MS precisa estar apta a acessar a rede GPRS.
BTS	Atualização de software nas BTS's existentes.
BSC	Atualização de software nos BSC's existentes e instalação de um novo hardware (PCU) para controle de tráfego de dados.
SGSN e GGSN	Dois novos elementos de rede para o serviço GPRS.
VLR, HLR, AuC, EIR	Atualização de software para suportar novas funções do GPRS.

A nova arquitetura necessita de uma PCU (*Packet Control Unit*), que disponibiliza interfaces lógicas e físicas para transmissão de dados. Com isso, o tráfego de dados é realizado do BSS até o SGSN (*Serving GPRS Support Node*), um novo elemento adicionado na rede. A tráfego de voz continua sendo realizado através da arquitetura GSM, ou seja, do BSS para o MSC. Em resumo, o SGSN provê acesso das MSs à rede GPRS. (SVERZUT, 2015)

Outro elemento presente na nova arquitetura de rede é o GGSN (*Gateway GPRS Support Node*). Com ele é possível realizar a conexão da rede GPRS com redes externas. Também, foi incluso na rede o DNS (*Domain Name Service*), cuja função é transformar endereços IPs em nomes de domínio, assim facilitando a navegação. O DHCP (*Dynamic Host*

Configuration Protocol) é outra novidade na tecnologia GPRS, onde é responsável por distribuir dinamicamente os endereços IPs e, também, foram adicionados dispositivos de segurança como *Firewalls*. Por fim, foi inserido o serviço APN (*Access Point Name*), onde são armazenados endereços IP em cada interface externa de rede conectada ao GGSN. A Figura 4 ilustra a arquitetura GPRS. (PIROTTI; ZUCCOLOTTO, 2009)

Figura 4 – Arquitetura GPRS.



Fonte: (SVERZUT, 2015)

2.2 Rastreadores

Esta Seção aborda os conceitos e tipos de rastreadores veiculares.

2.2.1 Rastreamento veicular

Através do uso de módulos de rastreamento móvel, é possível saber a localização de qualquer tipo de veículo. Desta maneira, se obtém uma sensação de maior segurança, visto que os índices de furtos de veículos só aumentam em nosso país. Na maioria dos casos este rastreamento é feito com a instalação de um pequeno dispositivo em alguma parte pouco visível do veículo, para emissão de ondas através de antenas. (MANTUANO, 2018)

A melhora na segurança é fator muito relevante, pois, de acordo com (ELAINA, 2019), a cada dez veículos roubados com rastreador, nove são recuperados. Em um primeiro momento, a instalação desse dispositivo é algo que gera um determinado custo financeiro. Porém, ele pode gerar ganhos imensuráveis a longo prazo, como maior tranquilidade ao se locomover, mais segurança para demais passageiros e, também, é possível conseguir descontos na aquisição de um seguro para o veículo. Os descontos com carros que possuem rastreadores podem chegar até a um quarto do valor, dependendo do perfil de usuário e da política da seguradora. De acordo com Fábio Braga, superintendente da Porto Seguro Proteção e Monitoramento: "Se o perfil do motorista for considerado 'ruim' para um seguro tradicional, o seguro com rastreador tende a ser mais barato". (ALMEIDA, 2017)

2.2.1.1 Tipos de Rastreadores Veiculares

Atualmente, existem diferentes tipos de rastreadores veiculares em operação; os que utilizam GPS e os sistemas de radiofrequência. Visto isso, os rastreadores que usam GPS optam por uma rede de que possua no mínimo quatro satélites e são capazes de emitir a localização precisa através de coordenadas de latitude e longitude. Enquanto os rastreadores que utilizam a tecnologia de radiofrequência captam o sinal através de uma triangulação de sinal. (MANTUANO, 2018)

Existe, também, o tipo GSM/GPRS, que pode ser definido pela junção das duas tecnologias anteriormente citadas. A transmissão de dados é realizada através de um chip GSM e o mesmo recebe as informações do GPS. A comunicação entre este dispositivo e o seu servidor é customizável para cada caso. Porém, de uma maneira genérica, é possível afirmar que os dados são enviados para um servidor que processa a informação, assim disponibilizando para cada plataforma específica utilizar. (MANTUANO, 2018)

Não existe um método bem definido para a escolha do melhor tipo de rastreador. Isso é algo que depende de muitas situações. Para escolher um rastreador via radiofrequência, é necessário que o veículo trafegue a maior parte do tempo em centros urbanos. Isto porque possuem boa cobertura de sinal e é possível obter a localização em locais fechados, sendo suscetível a poucas interferências externas capazes de atenuar o sinal. A escolha do rastreador via GPS deve ocorrer quando o veículo irá percorrer a maior parte do percurso em locais mais remotos, com longas distâncias, pois é possível ter pouca cobertura da radiofrequência. (MANTUANO, 2018)

2.3 Arduino

Nesta Seção será abordado o Arduino, apresentando seus conceitos, mostrando suas origens, os exemplos de modelos, a IDE e suas aplicabilidades.

2.3.1 O que é um Arduino?

Segundo (MCROBERTS, 2011), pode se dizer que um Arduino é um computador de pequeno porte, que pode ser programado para processar entradas e saídas do próprio dispositivo ou de componentes externos conectados. Também, é possível dizer que o Arduino é uma plataforma de computação embarcada. Sendo assim, interagindo com *hardware* e *software*. Dito isto, o Arduino pode ser utilizado em diversos tipos e tamanhos de projetos, sendo eles independentes ou conectados com um computador e podendo utilizar a internet para transferência de dados.

2.3.2 Origem do Arduino

O Arduino foi criado em 2005, na Itália, pelos pesquisadores Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. O projeto foi criado com o propósito de oferecer uma ferramenta de prototipagem eletrônica *open-source*, que fosse fácil o aprendizado para o desenvolvimento. O sucesso do projeto foi muito grande, sendo citado em menções honrosas em diversos congressos e tendo mais de 50 mil placas vendidas nos primeiros três anos. (MOTA, 2017)

Com isso, surgiu uma placa que possuía um microcontrolador, com vários circuítos de entrada e de saída simulando o funcionamento de um computador, obviamente, de forma muito simplória. A parte de programação é feita através de uma IDE que os fabricantes da plataforma disponibilizam, sendo necessário apenas um cabo USB tipo A/B para conectar a placa com o computador onde o *softwate* está. (THOMSEN, 2014b)

2.3.3 Exemplos de modelos

Nesta Seção serão abordados os seguintes exemplos de modelos de arduino: Uno, Mega, Mini e Nano.

2.3.3.1 Arduino Uno

O Arduino Uno caracteriza-se por ser a placa inicial no aprendizado dessa ferramenta, visto a sua simplicidade e sua facilidade de ser utilizada juntamente com *shields* (Circuitos que podem ser acoplados diretamente ao Arduino). Com isso é possível ampliar e diversificar as funções desenvolvidas. (TAVARES, 2018)

A sua alimentação pode ser realizada através de USB, onde a tensão correspondente é de 5V, e através de um carregador externo, podendo variar a tensão entre 7V e 12V, devido ao seu regulador de tensão interno. Falando de suas conexões, o Arduino Uno possui 6 pinos analógicos (podem assumir infinitos estados de tensão) de 10 bits, 6 pinos com capacidade PWM (Modulação por largura de pulso) e 14 pinos digitais (podem apenas trabalhar em dois estados lógicos: com 0V ou 5V). Além disso, possui uma memória flash de 32 KB, SRAM

(Memória estática de acesso aleatório) de 2 KB, uma EEPROM (Memória não-volátil usada em dispositivos eletrônicos para armazenar pequenas quantidades de dados que precisam ser salvos quando não há energia) de 1 KB e microcontrolador ATmega328P. A Figura 5 ilustra o Arduino Uno. (TAVARES, 2018)

Figura 5 – Exemplo do Arduino Uno.

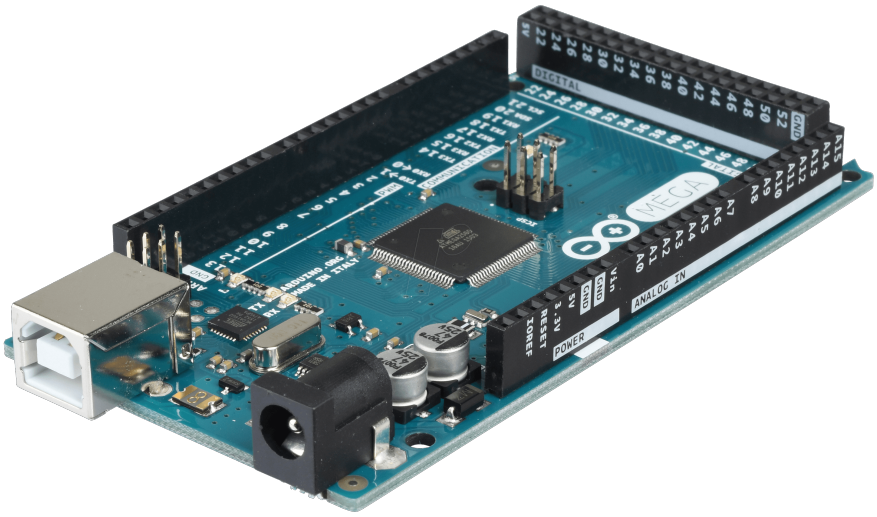
Fonte: (TAVARES, 2018)

2.3.3.2 Arduino Mega

O Arduino Mega é uma versão mais robusta que o citado anteriormente, o Arduino Uno. Isto porque possui um número maior de pinos. Com isso, se um *shield* for acoplado ao Arduino Mega não serão todos pinos utilizados, como acontece no Arduino Uno. Também possui maior quantidade de memória *flash*. (TAVARES, 2018)

Em termos das características do Arduino Mega, possui 54 pinos digitais (sendo 15 que podem ser usados para PWM) e 16 pinos analógicos. Além disso, contém memória *flash* de 265KB, memória SRAM de 8KB e memória EEPROM com 4KB. A Figura 6 mostra o Arduino Mega. (TAVARES, 2018)

Figura 6 – Exemplo do Arduino Mega.

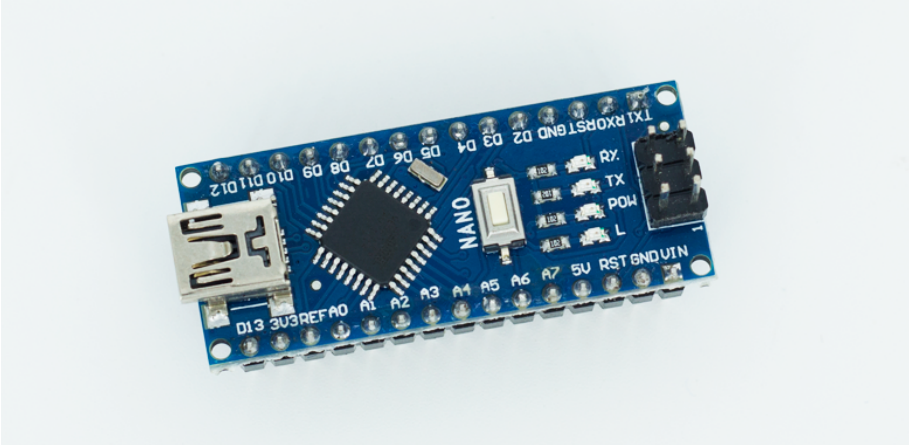


Fonte: (TAVARES, 2018)

2.3.3.3 Arduino Nano

O Arduino Nano possui muitas semelhanças com o Arduino Uno. A sua grande vantagem em relação aos demais é o seu tamanho, que corresponde a apenas 4,3 centímetros de comprimento e 1,85 centímetros de largura. Com isso, é possível usá-lo em pequenos espaços. Também é importante mencionar que o Arduino Nano não é alimentado por uma fonte externa, e sim por um USB Mini-B. A Figura 7 apresenta o Arduino Nano. (THOMSEN, 2014c)

Figura 7 – Exemplo do Arduino Nano



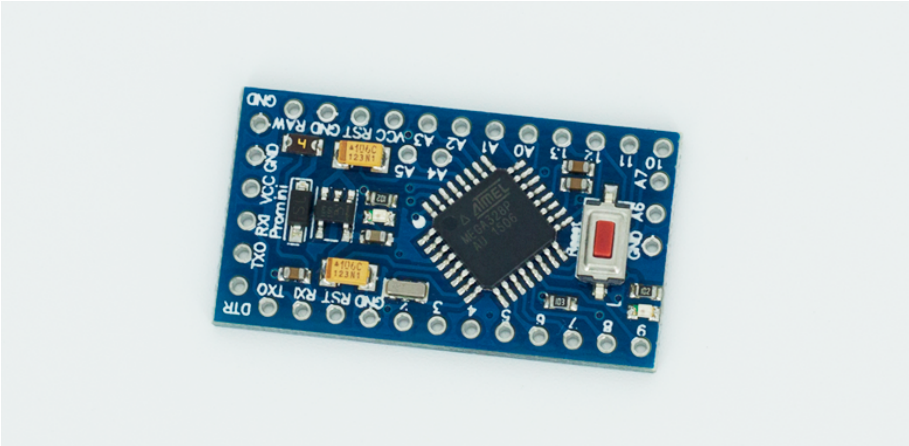
Fonte: (THOMSEN, 2014c)

2.3.3.4 Arduino Mini

O Arduino Mini possui diversas semelhanças com o Arduino Uno e o Arduino Nano, apresentando dimensões menores que a versão Nano. As principais diferenças são que contém

versões com microcontrolador ATmega168 (possui memória de programação de 16 KB e memória EEPROM com 512 *bytes*) e ATmega328P (possui memória de programação de 32 KB e memória EEPROM com 1024 *bytes*) e possui *clock* (relógio de tempo real com calendário completo) de 16 e 8 MHz. Possui uma quantidade menor de pinos, sendo encontrado apenas 4 analógicos e necessita ser alimentado com um cabo especial FTDI (*Future Technology Devices International*). A Figura 8 representa o Arduino Mini. (THOMSEN, 2014c)

Figura 8 – Exemplo do Arduino Mini.



Fonte: (THOMSEN, 2014c)

2.3.4 Arduino IDE

O Arduino IDE é uma ferramenta desenvolvida na linguagem JAVA, que disponibiliza um ambiente completo para o desenvolvimento de software embarcado para Arduíno. Com a aplicação é possível compilar códigos de forma muito simples. (THOMSEN, 2014b)

A linguagem utilizada para escrever as instruções do Arduíno é baseada em C e C++, porém não é necessário possuir um conhecimento profundo nessas tecnologias para começar a desenvolver. A estrutura do código basicamente é dividida em uma função *setup()*, que é responsável por realizar as configurações iniciais do programa e a função *loop()*, que é onde a estrutura principal do código é executada em forma de repetição até que algum comando pare o processo. (THOMSEN, 2014b)

Segue a Figura 9, com exemplo de um pequeno código, chamado de *Blink*, que tem por objetivo apagar e acender uma luz de LED.

Figura 9 – Exemplo de código arduino.

```
//Programa : Pisca Led Arduino
//Autor : FILIPEFLOP

void setup()
{
  //Define a porta do led como saida
  pinMode(13, OUTPUT);
}

void loop()
{
  //Acende o led
  digitalWrite(13, HIGH);

  //Aguarda o intervalo especificado
  delay(1000);

  //Apaga o led
  digitalWrite(13, LOW);

  //Aguarda o intervalo especificado
  delay(1000);
}
```

Fonte: (THOMSEN, 2014b)

2.3.5 Aplicabilidade

Como já foi dito anteriormente, o principal objetivo do Arduíno é facilitar a prototipagem de sistemas eletrônicos. Com ele é possível automatizar processos em diversos níveis, tais como, no ambiente doméstico, ambientes empresarias, no âmbito industrial, na aprendizagem de robótica escolar entre outros. Um bom exemplo para isso é a utilização de sensores. Com eles é possível ligar ou apagar luzes de acordo com a hora e presença de pessoas, também se consegue abrir e fechar janelas de acordo com a temperatura do ambiente e da umidade. Em resumo, com a facilidade de integração de outros dispositivos a criatividade acaba se tornando o limite de uma aplicação utilizando Arduíno. (THOMSEN, 2014b)

2.4 Aplicativos Móveis

Nesta Seção será demonstrado características dos aplicativos móveis, dados estatísticos e técnicas de desenvolvimento.

2.4.1 Conceitos básicos sobre aplicativos móveis

Aplicação móvel ou aplicativo móvel, sendo usualmente denominado por *app*, consiste em um software que pode ser executado em um dispositivo móvel, sendo o mais comum um *smartphone*. Os aplicativos móveis podem ser instalados manualmente no dispositivo pelo

usuário, como podem, também, já serem instalados no aparelho como configuração de fábrica. Porém, na maioria dos casos, são adquiridos através das lojas de cada plataforma, sendo as mais conhecidas *Google Play* e *Apple Store*, que disponibilizam, respectivamente, aplicativos para o sistema operacional Android e IOS (Sistema operacional móvel da *Apple*). (SUMARES, 2017)

Os *apps* são disponibilizados de forma gratuita e também de forma paga. Porém, existem muitos aplicativos que são uma mistura dos dois, isto é, possuem uma versão inicial com poucas funcionalidades de forma gratuita, e se o usuário desejar pode comprar a versão completa do aplicativo.

2.4.2 Dados estatísticos sobre aplicativos móveis

O número de *downloads* de aplicativos estão em constante aumento, chegando a 113 bilhões em 2018 e, com isso, aumentando 10% em relação a 2017. A partir deste fato, é possível perceber que a tecnologia está se consolidando no segmento dos dispositivos móveis, o que já não é novidade há vários anos. (ROSA, 2018)

O lado financeiro do mercado de aplicativos móveis também está em forte ascensão, visto que, obteve-se um aumento de 20% de 2017 para 2018, alcançando o patamar de 76 bilhões de dólares movimentados. (ROSA, 2018)

Trazendo os números para a realidade brasileira, estima-se que o número de usuários de *smartphones* no Brasil, em 2016, chegou à incrível marca de 168 milhões de pessoa, representando cerca de 80% da população. Além disso, dados de 2018 apontam que o número de aparelhos celulares no Brasil ultrapassou o número de habitantes, chegando a 220 milhões. (DEMARTINI, 2018)

Em nível mundial, as estatísticas apontam para aproximadamente 2 bilhões de pessoas, projetando que chegue a 3 bilhões até 2020. Levando em conta o consumo do tráfego da internet, cerca de 36% é realizado via *smartphones* em 2016 e chegando a 45% em 2017. Em 2014, o número de usuários de *smartphones* ultrapassou o número de usuários de *desktops*, consolidando os dispositivos móveis como a maior tendência do mercado eletrônico. (MOURãO, 2017)

2.4.3 Desenvolvimento de Aplicativos Móveis

O desenvolvimento de aplicativos móveis varia a sua complexidade de acordo com a experiência dos programadores envolvidos e a escolha das ferramentas corretas, observando, que cada projeto tem seus requisitos próprios e particulares. Tendo isso em mente, é necessário que o desenvolvedor possua uma boa lógica de programação, domínio de uma linguagem adequada para o projeto proposto e boas noções de documentação e modelagem de aplicações. (SCUDERO, 2017)

2.4.3.1 Aplicativos nativos

Os denominados aplicativos nativos são desenvolvidos com linguagens de programação direcionadas exclusivamente para cada sistema operacional. Com isso, é possível acessar as funcionalidades do dispositivo como câmera, GPS, entre outras, de uma forma direta, ou seja, sem depender de uma camada intermediária para isso. Uma das grandes vantagens dos aplicativos nativos é um significativo ganho de desempenho, isto porque utilizando os recursos próprios de cada sistema operacional é possível lapidar o projeto da melhor forma. O lado negativo do desenvolvimento nativo é o aplicativo não funcionar em outra plataforma, elevando o custo de programação e, consequentemente, aumentando o valor do produto. Bons exemplos de aplicativos nativos no mercado são *Facebook* e *Whatsapp*. (SCUDERO, 2017)

Ao desenvolver para cada plataforma específica, é necessário escolher uma linguagem condizente. Se o programador optar em desenvolver para IOS, é possível escolher entre *Objective-C* ou *Swift*. Não existe uma linguagem perfeita genérica, cada projeto possui os seus requisitos e a partir deles é feita a melhor escolha. Vale ressaltar que *Swift* é considerada a sucessora de Objective-C, sendo mais moderna e recente. A plataforma Android é a mais popular e a que mais emprega atualmente, sendo JAVA a linguagem majoritária. (SCUDERO, 2017)

Nesse contexto, o *React Native* surgiu como uma quebra de paradigma. Pois, com ele, não é mais necessário ter uma linguagem em cada plataforma para desenvolver aplicativos nativos. Com o *React Native* é possível desenvolver o aplicativo usando apenas *JavaScript* e gerar um produto tanto para Android quanto para IOS. A ferramenta de forma interna consegue traduzir o código e fazer chamadas nativas para cada sistema operacional e, ainda, abstraindo esse processo do entendimento do programador, deixando com ele apenas a responsabilidade de manipular *JavaScript*. O *React Native* foi criado pelo *Facebook* em 2015. (CâMARA, 2018)

2.4.3.2 *Web Apps*

Os chamados *Web Apps*, na prática, não podem ser considerados aplicativos de forma verídica. Isto porque eles são aplicações web que estão preparados para serem executadas em dispositivos móveis, e são desenvolvidos com tecnologias web como, na maiorias dos casos, HTML, CSS e JavaScript. Um ponto positivo desse método é que os *web apps* podem ser executados em qualquer sistema operacional, com a ressalva de possuir a necessidade do dispositivo estar conectado com a internet, para realizar requisições a um servidor web. Na maioria dos casos são menos performáticos que os aplicativos nativos, porém possuem menor complexidade para o seu desenvolvimento. (MADUREIRA, 2017)

2.4.3.3 Aplicativos híbridos

Os aplicativos híbridos são desenvolvidos utilizando tecnologias que são usadas nos *web apps*, sendo elas HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) e *JavaScript*, e são embutidos em um código que está relacionado com as funcionalidades nativas do dispositivo. Com isso, é possível distribuir o aplicativo para mais de uma plataforma, sendo necessário apenas reescrever o código correspondente à parte nativa. Esta técnica facilita a atualização do aplicativo, pois não é necessário fazer *download*, porém ainda continua sendo necessária a conexão com internet para o funcionamento. (MADUREIRA, 2017)

3 DESENVOLVIMENTO

Nesta Capítulo serão explanadas as ferramentas utilizadas no desenvolvimento do projeto do trabalho e, também, descrito toda a parte do desenvolvimento do sistema embarcado em questão, da API e da aplicação móvel.

3.1 Ferramentas Utilizadas

Antes de começar o tópico é importante frisar as ferramentas Arduino IDE e Arduino Uno foram explanadas no Capítulo 2, por isso, não serão aprofundadas.

3.1.1 GPRS/GSM *Shield* SIM900

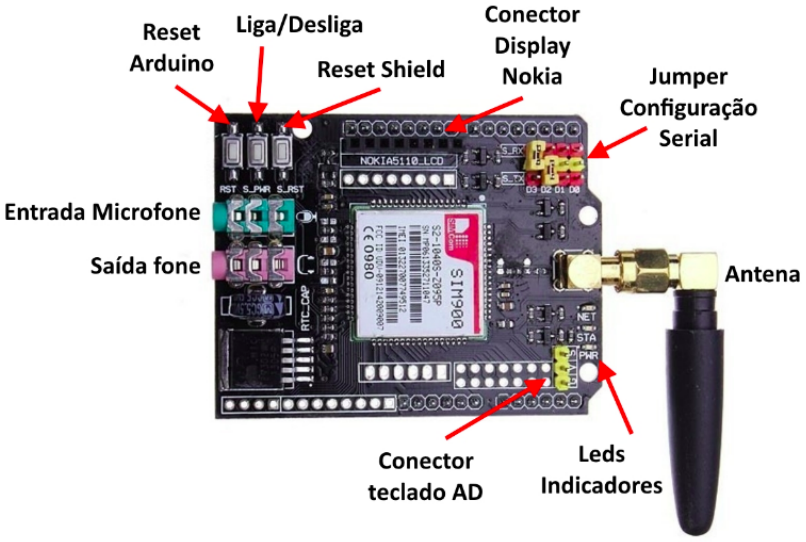
Foi desenvolvida pela empresa chinesa SIMCom. O embarcado em questão é um módulo sem fio SIM900 Quad-Band (850/900/1800/1900 Mhz) GSM/GPRS. Com ele é possível obter várias funções de um aparelho celular, tais como enviar mensagem de texto (SMS), fazer ligações, trafegar dados via internet e obter informações de localização. Pode ser utilizada com qualquer arduino que possui pinagem padrão, por exemplo Uno e Mega. Além disso, possui um baixo consumo de energia. (THOMSEN, 2014a)

3.1.1.1 Estrutura da placa

Falando sobre a estrutura da placa, na parte central fica localizado o chip SIM900, que faz o gerenciamento da placa. Além disso, possui entradas para microfone e saída de áudio, através de conectores de 3,5mm. Também, é possível fazer a instalação de um LCD Nokia 5110 (versão 3.3v), para visualização de informações e também de um teclado. Para melhorar o sinal, possui um conector para antena externa e na parte traseira que também possui a entrada para o cartão SIM. (THOMSEN, 2014a)

Segue Figura 10 e Figura 11 ilustrando do que foi citado anteriormente:

Figura 10 – Estrutura GPRS/GSM *Shield* SIM900.



Fonte: (THOMSEN, 2014a)

Parte traseira:

Figura 11 – Estrutura traseira GPRS/GSM *Shield* SIM900.

Fonte: (THOMSEN, 2014a)

Para um bom funcionamento, é recomendado, pelo próprio fabricante, que se utilize uma fonte 9V 1A externa, pois a alimentação apenas da porta USB não é suficiente para manter o *shield* em bom funcionamento.

Este *shield* é configurado através de comandos AT, que são inseridos no *monitor serial*. Segue alguns exemplos de comandos que podem ser utilizados. (THOMSEN, 2014a)

- AT - Retorna “OK” se a comunicação com o shield estiver OK;
- A/ - Reenvia o último comando;
- ATD NUMERO - Efetua uma ligação para o telefone especificado em NUMERO;
- ATDL - Repete a última ligação feita;
- ATZ - Carrega as configurações padrão;
- AT&F - Retorna às configurações de fábrica;
- AT+CSQ - Mostra a qualidade do sinal.

3.1.2 Linguagem de programação *Ruby*

O *Ruby* é uma linguagem de programação interpretada, que foi lançada em 1995, no Japão, por Yukihiro Matsumoto, popularmente conhecido como Matz, porém já havia sido criada desde 1993. Matz inicialmente buscava uma linguagem de *script* melhor que *Perl* e orientada a objetos como *Python*. Depois disso, o *Ruby* evoluiu e é capaz de manter diversos paradigmas como programação funcional, orientada a objetos, imperativa e reflexiva. (SOUZA, 2014)

3.1.2.1 Tipo de dados

A classe abstrata *Numeric* é uma classe que representa os números, sendo herdada pelas classes *Integer*, representa os valores inteiros, e *Float*, que representa valores reais. Os tipos *Fixnum* (possui tamanho fixo) e *Bignum* (tamanho ilimitado, de acordo com a memória disponível) são derivados da classe pai *Integer*. (SILVA, 2015)

Falando sobre *Strings*, são cadeias de caracteres usadas para representar textos na aplicação. São definidas de duas formas: com apas simples (’), onde o texto é interpretado literalmente e com aspas duplas ("), onde pode conter símbolos e expressões que podem ser interpretados. (SILVA, 2015)

Outros tipos de dados mais usados são *Arrays* e *Hashs*, que podem certa semelhança, pois ambos armazenam coleções de dados ou elementos não sendo obrigatórios serem de mesmo tipo. A principal diferença é que o *hash* possui índice em cada elemento. Segue exemplos. (SILVA, 2015)

```
nome_da_array = [ elemento1, elemento2, elemento3 ]
```

```
nome_da_hash = { objeto_chave1 => valor1, objeto_chave2 => valor2 }
```

3.1.3 *Ruby On Rails*

Esta Seção abordará o *Ruby On Rails*, sua história, arquitetura, convenções e principais componentes.

3.1.3.1 História

O *framework Ruby On Rails* foi lançado ao público em 2003 por David Heinemeier Hansson. A sua criação ocorreu enquanto David trabalhava no projeto *Basecamp*, uma ferramenta de gerenciamento de projetos. Em 2004 a ferramenta se tornou *open source*, o que permitiu a contribuição de diversos desenvolvedores, assim evoluindo de maneira significativa. (SCORZA, 2016; MCWHAE, 2019)

As versões seguintes do *Rails* foram lançadas nas seguintes datas:

- *Rails* 1.0: Dezembro 2005
- *Rails* 2.0: Dezembro 2007
- *Rails* 3.0: Agosto 2010
- *Rails* 4.0: Junho 2013
- *Rails* 5.0: Julho 2016
- *Rails* 6.0: Agosto 2019

3.1.3.2 Arquitetura

Projetos desenvolvidos com *Ruby On Rails* possuem a arquitetura MVC (*Model-View-Controller*). Onde o *Model* é responsável pela leitura e escrita de dados, o *View* tem a função de apresentação de dados e o *Controller* gerencia as requisições do usuário.

3.1.3.3 Convenções do *Ruby On Rails*

Uma das principais convenções adotas na comunidade do *Rails* é o DRY (*Don't Repeat Yourself*), em português, não se repita. Isso basicamente significa que é sempre preferencial reaproveitar códigos no projeto e evitar ao máximo redundância de código. (FUENTES, 2012)

Outra convenção relevante é a *Convention Over Configuration*, ou convenção à configuração, que define uma configuração prévia do projeto padrão, sendo alterada somente se necessário. Com isso, é muito fácil desenvolvedores migrarem de um projeto para outro, por a estrutura de configuração ser similar. (FUENTES, 2012)

3.1.3.4 Principais componentes do *Ruby On Rails*

Esta Subseção mostrará os principais componentes do *Ruby On Rails*.

- *Active Record*: é uma biblioteca que faz parte do *Model* na arquitetura MVC e trabalha realizando o mapeamento objeto-relacional, ou seja, transformando estruturas relacionais em objetos *Ruby*. O *Active Record* utiliza por baixo dos panos a biblioteca *Arel*, que é capaz de transformar métodos *Ruby* em consultas SQL e mapear em objetos. (FUENTES, 2012)
  Exemplo do funcionamento do *Active Record*:

```
Room.where(:location => ['São Paulo', 'Rio de Janeiro'])
    # SELECT "rooms".* FROM "rooms" WHERE "rooms"."location"
        # IN ('São Paulo', 'Rio de Janeiro')
```

- O *Action Pack* é dividido em dois componentes:
  - *Action Controller*: corresponde ao *Controller* no modelo MVC. É responsável por manipular o fluxo de dados a serem renderizados na visualização. (FUENTES, 2012)
  - *Action View*: corresponde ao *View* no modelo MVC. Tem a função de gerar os arquivos de visualização do projeto.(FUENTES, 2012)
- *Action Mailer*: é uma ferramenta responsável pelo gerenciamento do envio e recebimento de *e-mail*. Trata-se de uma ferramenta simples, porém muito poderosa para ealizar operações referentes a entregas de correspondências eletrônicas. (FUENTES, 2012)

3.1.4 *PostgreSQL*

PostgreSQL é um sistema gerenciador de banco de dados objeto relacional, que possui código aberto. Seu desenvolvimento ocorreu inicialmente na Universidade de Berkeley na Califórnia em 1982, através de *Michael Stonebraker*. (POSTGRESQL, 2019)

Os seus principais recursos são: consultas, chaves estrangeiras, integridade transacional, controle de concorrência, multi-versão, suporte ao modelo híbrido objeto-relacional, facilidade de acesso, gatilhos, visões, linguagem procedural, indexação por texto entre outras. (POSTGRESQL, 2019)

3.1.5 *React Native*

O *React Native* é um *framework* desenvolvido e mantido pelo *Facebook*, lançado no ano de 2015. Tem como objetivo fornecer estrutura para o desenvolvimento de aplicativos nativos para as plataformas *Android* e IOS. Seu lema é *learn once, write anywhere*, ou seja, aprenda uma vez, escreva em qualquer lugar. Atualmente se encontra na versão 0.61. (ZANDAVALLE; SILVA, 2018; FACEBOOK, 2019)

O *React Native* utiliza a linguagem de programação *JavaScript*, com a sintaxe JSX (*JavaScript eXtension*), onde é possível manipular os elementos em telas através de componentes, como acontece no *ReactJS*. Após ser escrito pelos desenvolvedores, ele é executado diretamente no dispositivo através de serialização. Estas características agregam muito o desenvolvimento na questão de performance. (ZANDAVALLE; SILVA, 2018; FACEBOOK, 2019)

Segundo (ZANDAVALLE; SILVA, 2018):

> Essas características e aspectos apresentados, podem ser a razão de grandes empresas e "startups" como Facebook, Instagram, Skype, Uber, Walmart, entre tantas outras, estarem usando react native para a construção de seus aplicativos de acordo com o site oficial da ferramenta. Além disso, por se tratar de um framework que utiliza uma biblioteca para criação de interfaces para usuário já muito utilizada no mercado e desenvolvida pela mesma empresa, como o react, não exige que o desenvolvedor tenha a necessidade de aprender duas novas linguagem como Objective-C (IOS) e java(Android) para o seu desenvolvimento, podendo ainda o código ser amplamente re-aproveitado para as duas plataformas.

O código a seguir apresenta a sintaxe básica do *React Native*.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';
export default class HelloWorldApp extends Component {
  render() {
    return (
      <View>
        <Text>Hello, world!</Text>
      </View>
    );
  }
}
```

Dito isto, o *React Native* foi escolhido para ser usado como ferramenta do desenvolvimento do aplicativo do presente trabalho, isto por ser uma plataforma que facilita o desenvolvimento e ainda propícia uma ótima performance.

3.1.6 *Google Maps API*

O *Google* disponibiliza através de sua API vários serviços de mapas e imagens de satélite que podem ser implementados em diversos softwares, de forma gratuita mas com limite de consumo. Os principais serviços oferecidos são os seguintes: (GOOGLE, 2019)

- *Maps*: Mostra todo mundo em mapas, tendo 99% de cobertura e 25 milhões de atualizações por dia. Disponibiliza imagens de *Street View*, onde é possível visualizar ruas com imagens de alta resolução. (GOOGLE, 2019)
- *Routes*: Fornece aos usuários trajetos ótimos entre dois o mais pontos em um mapa, além de informações de trânsito em tempo real. (GOOGLE, 2019)
- *Places*: Oferece dados sobre cerca de 100 milhões de locais, permitindo localizá-los através de endereços, números de telefone entre outros. (GOOGLE, 2019)

A Figura 12 irá demonstrar o funcionamento do *Google Maps*:

Figura 12 – Exemplo *Google Maps*

Fonte: (GOOGLE, 2019)

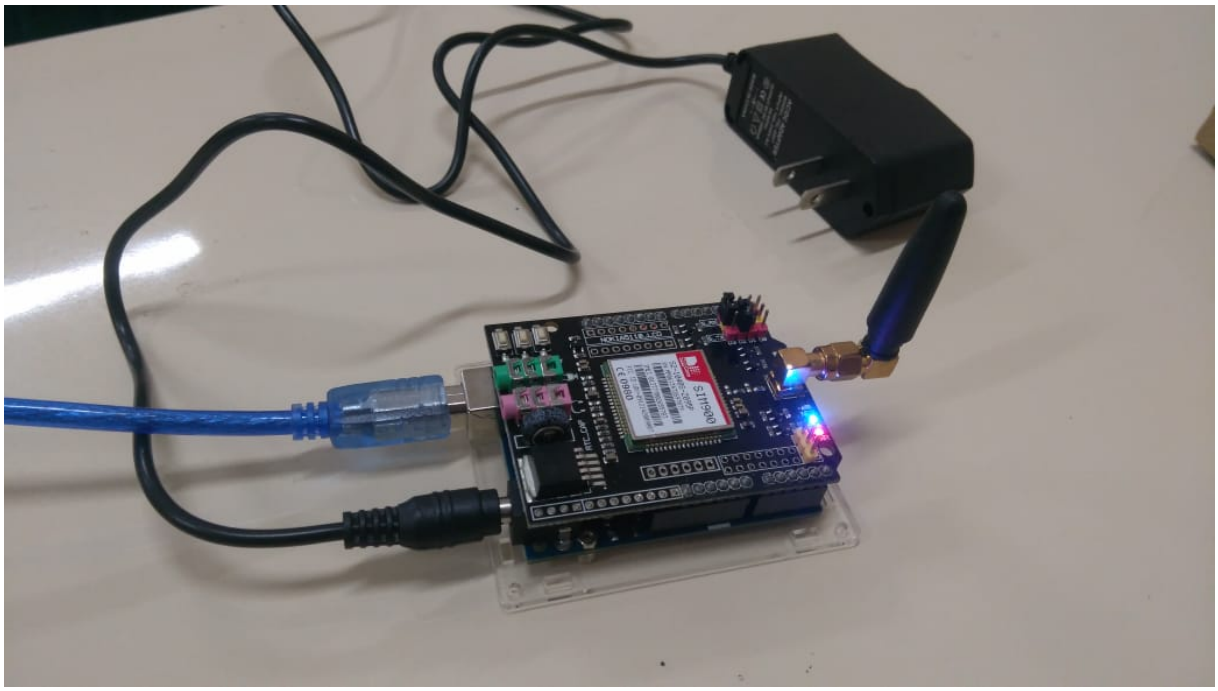
A API do *Google Maps* foi escolhida para o desenvolvimento do trabalho por ser a ferramenta líder no quesito de mapas e pela facilidade na sua implementação no projeto *React Native*. A configuração completa da implementação da API será descrita detalhadamente na seção 6.3 no presente trabalho.

3.2 Montagem e configuração do módulo *GSM GPRS Shield EFCom SIM900*

Para coleta dos dados de localização foi desenvolvido um módulo composto por Arduino Uno e *GSM GPRS Shield EFCom SIM900*, sendo alimentado por uma fonte de energia externa de 9V. O módulo em questão possui tecnologia GSM/GPRS. Com isso é possível conectar-se a internet através da rede 4G, para realizar o envio dos dados ao servidor.

A Figura 13 irá demonstrar o módulo em funcionamento:

Figura 13 – Modulo GSM GPRS Shield EFCom SIM900 com Arduino Uno.



Fonte: AUTOR

Após inserido o cartão SIM, o módulo foi conectado ao computador via porta USB. Depois disso foi realizada a programação da placa, através da implementação de um código com base na linguagem C, na plataforma Arduino IDE. A primeira parte foi a preparação do ambiente através da importação da biblioteca *GSM-GPRS-GPS-Shield*, onde estão todas as funcionalidades disponíveis para o módulo SIM900.

Para realizar a coleta das informações de geolocalização foi configurado uma série de comandos AT responsáveis por ativar a conexão de internet móvel do módulo e o comando que retorna a latitude e longitude do veículo é:

```
AT+CIPGSMLOC=1,1
```

Após isso, foi armazenado em uma variável do tipo *string* os dados no formato correto, que posteriormente é utilizado na requisição para o envio dos dados na API. Para realizar a conexão com a internet foi utilizada a biblioteca *inetGSM.h*, que faz requisições do tipo *Get e Post*.

A Figura 14 demonstra a função *Setup()*, que é responsável por inicializar a placa e realizar as configurações inicias de conexão.

Figura 14 – Função *setup()*

```
void setup()
{
  //Inicializa a serial
  Serial.begin(9600);
  Serial.println("Testando GSM Shield...");
  //Inicia a configuracao do Shield
  if (gsm.begin(9600)){
    Serial.println("Módulo pronto!");
    gsm.SimpleWriteln("AT+CGATT=1");
    gsm.SimpleWriteln("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");
    gsm.SimpleWriteln("AT+SAPBR=3,1,\"APN\",\"zap.vivo.com.br\"");
    gsm.SimpleWriteln("AT+SAPBR=1,1");
    boolean notConnected = true;
    while (notConnected){
      if(inet.attachGPRS("zap.vivo.com.br", "vivo", "vivo")){
        Serial.println("Conectando...");
        notConnected = false;
      }
    }
    Serial.println("Conectado!");
  }else{
    Serial.println("Não conectado!");
  }

}
```

Fonte: Autor

A Figura 15 possui a implementação da função *loop()*, que captura as informações de coordenadas, após isso realiza o tratamento dos dados para o formato correto e realiza a requisição HTTPS *Post* no servidor com os dados.

Figura 15 – Função *loop()*

```
void loop()
{
  gsm.SimpleWriteln("AT+CIPGSMLOC=1,1");
  for(int i = 0 ; Serial.available() > 0 && i<200 ; i++) {
     location[i] = Serial.read();
  }
  substring(location, longitude, 14, 10);
  substring(location, latitude, 25, 10);

  strcpy(data, latitude);
  strcat(data, "longitude=");
  strcat(data, longitude);

 numdata = inet.httpPOST("busfinderapi.herokuapp.com", 80,
                         "/api/v1/coordinates", data, response, 500);
}
```

Fonte: Autor

3.3 Desenvolvimento da API

Esta seção irá descrever o desenvolvimento da API, sua estrutura, configuração e implementações.

3.3.1 Estrutura e configurações iniciais do projeto

Para realizar o desenvolvimento da API do projeto, foi selecionado o *framework Ruby On Rails*, que é baseado na linguagem *Ruby*. A API é do tipo REST (*Representational State Transfer*), e realizada a sua comunicação através de objetos JSON.

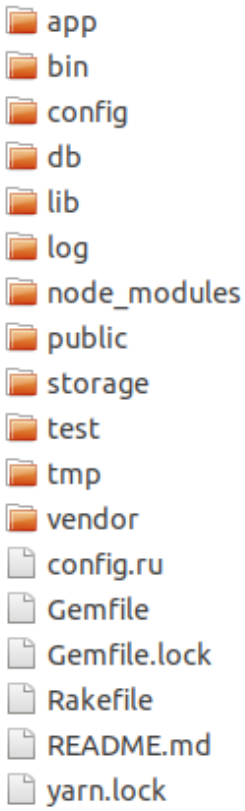
Para iniciar o projeto foi executado o comando:

```
rails new bus-finder-api --api --database=postgresql
```

Este comando é responsável por criar toda a estrutura inicial do projeto para uma API, e com a configuração para suportar o banco de dados *PostgreSQL*.

A estrutura de pastas gerada é ilustrada na Figura 16.

Figura 16 – Estrutura do projeto *Ruby on Rails*



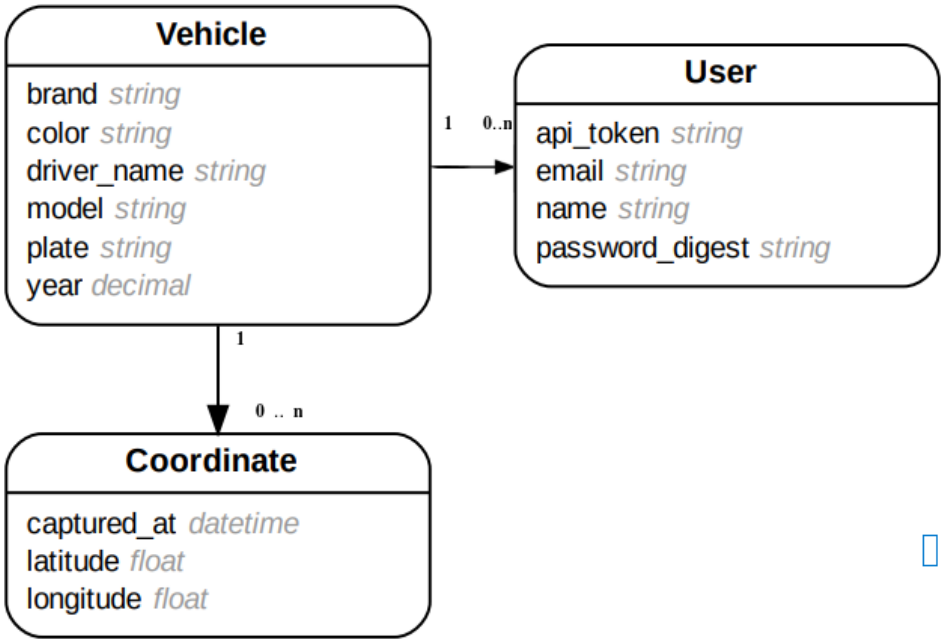
Fonte: Autor

Feito isso, para maior segurança e controle do projeto foi utilizada a ferramenta de versionamento de código *Git*, através da plataforma *GitHub*. A escolha do *GitHub* se deu pelo fato de a ferramenta ser amplamente difundida no mercado atual, oferecendo robustez e segurança para os desenvolvedores. Desta forma, todas versões do projeto se encontram no repositório *https://github.com/eliassartori/bus-finder-api*.

3.3.2 Modelo entidade-relacionamento

Com a estrutura inicial do projeto concluída, foram desenvolvidos os modelos de dados do projeto. As tabelas do projetos são as seguintes: Usuário, Veículo e Coordenadas. As tabelas podem ser visualizadas no modelo entidade-relacionamento na Figura 17.

Figura 17 – Modelo entidade-relacionamento



Fonte: Autor

3.3.3 Descrição de implementações do projeto

Após isso, foi implementado os métodos para inserir e ler as coordenadas do veículos na API. A Figura 18 descreve resumidamente o método para ler as coordenadas, através de requisições HTTPS *Get*.

Figura 18 – Função que retorna dados de coordenadas da API

```
# GET /coordinates
def index
  @coordinates = Coordinate.all.order("id DESC")

  render json: @coordinates
end
```

Fonte: Autor

Para o módulo GSM/GPRS SIM900 enviar dados ao servidor da API, a seguinte função foi criada para persistência de cada coordenada no banco de dados. A Figura 19 mostra de maneira simplificada o código responsável por realizar essa inserção via requisição *HTTPS Post*.

Figura 19 – Função que insere dados de coordenadas da API

```
# POST /coordinates
def create
  @coordinate = Coordinate.new(coordinate_params)

  if @coordinate.save
    render json: @coordinate, status: :created, location: @coordinate
  else
    render json: @coordinate.errors, status: :unprocessable_entity
  end
end
```

Fonte: Autor

Para acessar a API e inserir ou ler dados, é necessária uma autenticação. Esta autenticação é realizada através de uma chave de acesso por usuário. Cada vez que um novo usuário é criado no banco, ele recebe uma chave hexadecimal de 10 dígitos, e após isso, para cada requisição feita essa chave é solicitada.

A função para criação de um novo usuário está representada na Figura 20.

Figura 20 – Função que cria novo usuário na API

```
# POST /users
def create
  @user = User.new(user_params)

  if @user.save
    render json: @user, status: :created, location: @user
  else
    render json: @user.errors, status: :unprocessable_entity
  end
end
```

Fonte: Autor

O código responsável pelo modelo de dados que cria um novo usuário e atribui a ele a sua chave de acesso está ilustrado na Figura 21.

Figura 21 – Modelo que cria chave de acesso ao usuário

```ruby
class User < ApplicationRecord
    has_secure_password

    before_create :generate_api_token

    private

    def generate_api_token
        self.api_token = SecureRandom.hex(10)
    end
end
```

Fonte: Autor

Realizada a criação do usuário com sua chave de acesso, foi implementado na API uma verificação de segurança na realizações de requisições onde é necessário possuir o usuário autorizado. A implementação citada está representada na Figura 22.

Figura 22 – Autenticação na API

```ruby
class ApplicationController < ActionController::API
    before_action :authorize_access_request!

    protected

    def authorize_access_request!
        not_authorized unless current_user.present?
    end

    def not_authorized
        render json: { error: 'Not authorized' }, status: :unauthorized
    end

    def current_user
        api_token = request.headers['Authorization']

        return nil unless api_token

        User.find_by(api_token: api_token)
    rescue
        nil
    end
end
```

Fonte: Autor

Com a autenticação implementada, foi criado um controlador de sessão que é consumido pelo aplicativo móvel para realizar o *login*. Ele cria uma nova sessão na API se os dados de e-mail, senha e chave de acesso estiverem corretos. A Figura 23 ilustra o método de autenticação.

Figura 23 – Controlador de sessão na API

```
class Api::V1::SessionsController < ApplicationController
  skip_before_action :authorize_access_request!

  before_action :set_user, only: [:show, :update, :destroy]

  # POST /sessions
  def create
    @user = User.find_by(email: params[:email])

    if @user && @user.authenticate(params[:password])
      render json: { api_token: @user.slice(:name, :api_token) }, status: :authorized
    else
      render json: { error: 'E-mail ou senha inválidos' }, status: :unauthorized
    end
  end
end
```

Fonte: Autor

As implementações descritas anteriormente realizam os fluxos principais da API. Após toda a implementação ser finalizada, a API foi hospedada no serviço de nuvem *Heroku* que se encaixa na categoria Plataforma como Serviço (Platform as a Service, ou PaaS). O *endpoint* da API está disponível na seguinte URL de acesso: *https://busfinderapi.herokuapp.com/api/v1/*.

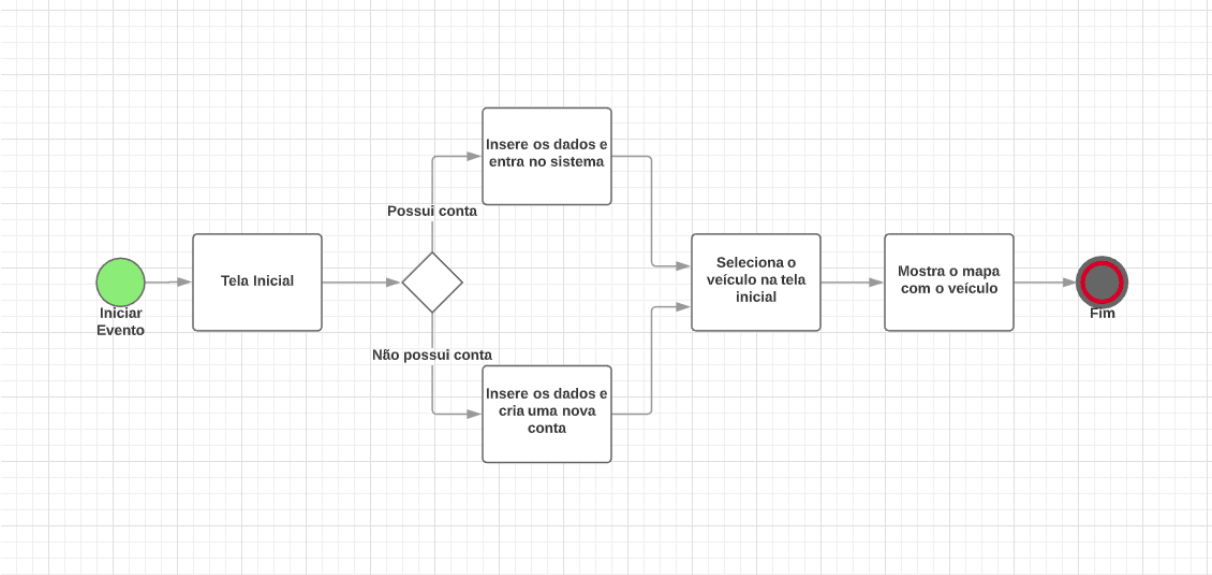
3.4 Desenvolvimento da Aplicação Móvel

Esta Seção descreve o desenvolvimento da aplicação móvel, com diagramas UML, estrutura inicial e descrição de implementações.

3.4.1 Modelagem UML

Foi criado um diagrama de atividade para modelar um caso onde o usuário faz o caminho natural entre telas do aplicativo. A Figura 24 representa esse diagrama.

Figura 24 – Diagrama de atividade



Fonte: Autor

3.4.2 Estrutura inicial do projeto

Para o desenvolvimento da aplicação móvel a ferramenta escolhida foi o *React Native*, que foi explanada no início deste Capítulo. Esta escolha se deu pela facilidade no desenvolvimento e pela robustez da plataforma, onde é possível criar aplicativos para mais de uma plataforma, sem alterações muito relevantes no código.

Para iniciar o projeto o seguinte comando foi executado:

```
react-native init BusFinder
```

3.4.3 Descrição de telas e implementações

Com a estrutura inicial do projeto pronta, foi desenvolvida a tela de entrada do aplicativo. A função responsável por fazer a criação da sessão na API está representada na Figura 25. A função faz uma requisição HTTPS *Post* na URL *https://busfinderapi.herokuapp.com/api/v1/sessions*, e, em caso de sucesso, retorna a chave de acesso.

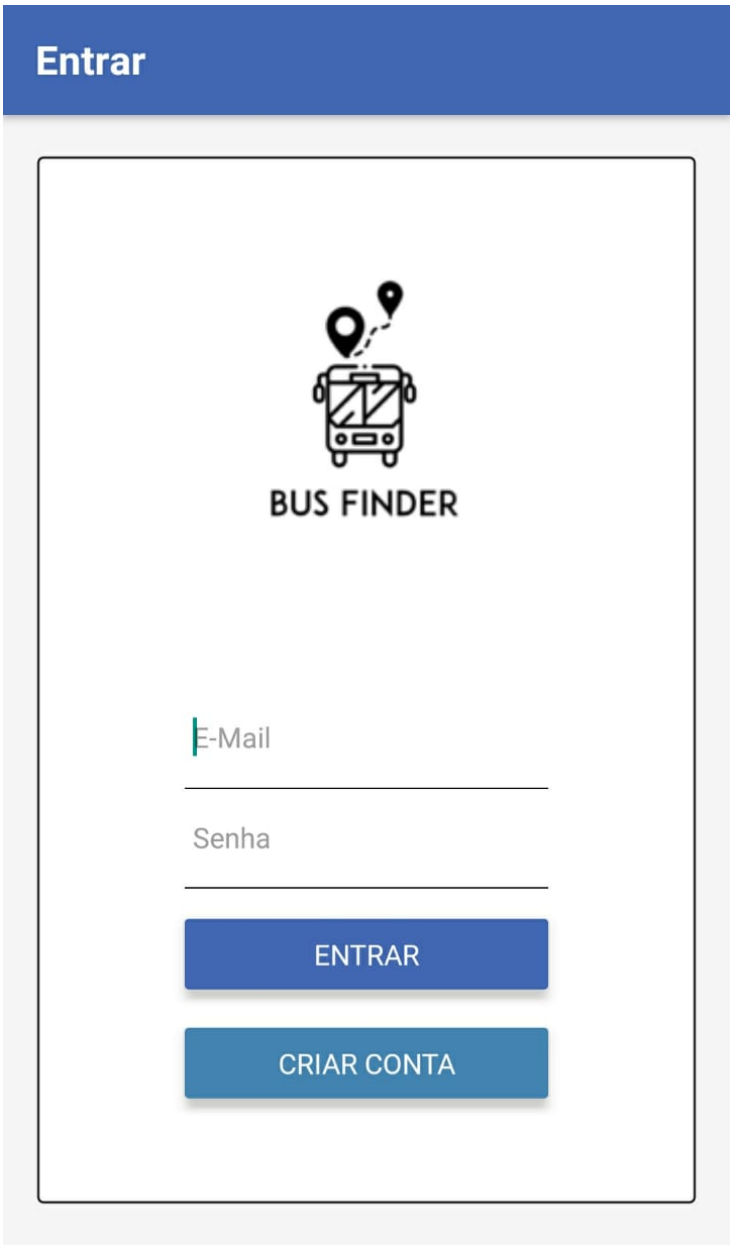
Figura 25 – Criação de sessão na API

```
const doLogin = () => {
  fetch(URL, {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      email,
      password,
    }),
  })
    .then(response => response.json())
    .then(data => {
      if (data) {
        if (data.error) {
          setError(data.error);
        } else {
          const api_token = data.api_token.api_token;
          navigation.navigate('index', {api_token});
        }
      }
    });
};
```

Fonte: Autor

A tela de entrada do aplicativo está ilustrada na Figura 26. Nessa tela o usuário pode informar seu *e-mail* e senha para entrar no aplicativo ou navegar para tela de criação de nova conta.

Figura 26 – Tela de entrada do aplicativo.



Fonte: Autor

Após a tela inicial criada, foi implementada uma tela para criação de novos usuários. A função que faz a criação de novo usuário está na Figura 27. Esta função realiza uma requisição na mesma URL da tela de entrada, *https://busfinderapi.herokuapp.com/api/v1/sessions* e, em caso de sucesso na criação, retorna para a tela de entrada.

Figura 27 – Função que cria novo usuário.

```
const createAccount = () => {
  if (password === passwordConfirm) {
    fetch(URL, {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        name,
        email,
        password,
      }),
    })
      .then(response => response.json())
      .then(data => {
        if (data) {
          if (data.error) {
            setError(data.error);
          } else {
            navigation.navigate('sign_in');
          }
        }
      });
  } else {
    setError('Senhas não correspondem');
  }
};
```

Fonte: Autor

A Figura 28 mostra como ficou a tela de criação de usuários, onde é necessário informar os dados para os campos Nome, *E-mail*, Senha e Confirmação de Senha.

Figura 28 – Tela de criação de conta.

Fonte: Autor

Após realizar a autenticação, o aplicativo vai para a tela de início onde o usuário tem os ônibus disponíveis para ele selecionar para rastrear. A Figura 29 apresenta a tela inicial.

Figura 29 – Tela inicial do aplicativo.

Fonte: Autor

Para exibir essa tela inicial, os dados são retornados da API, através de uma requisição a URL: *https://busfinderapi.herokuapp.com/api/v1/vehicles*. A função que busca esses dados segue na Figura 30 abaixo.

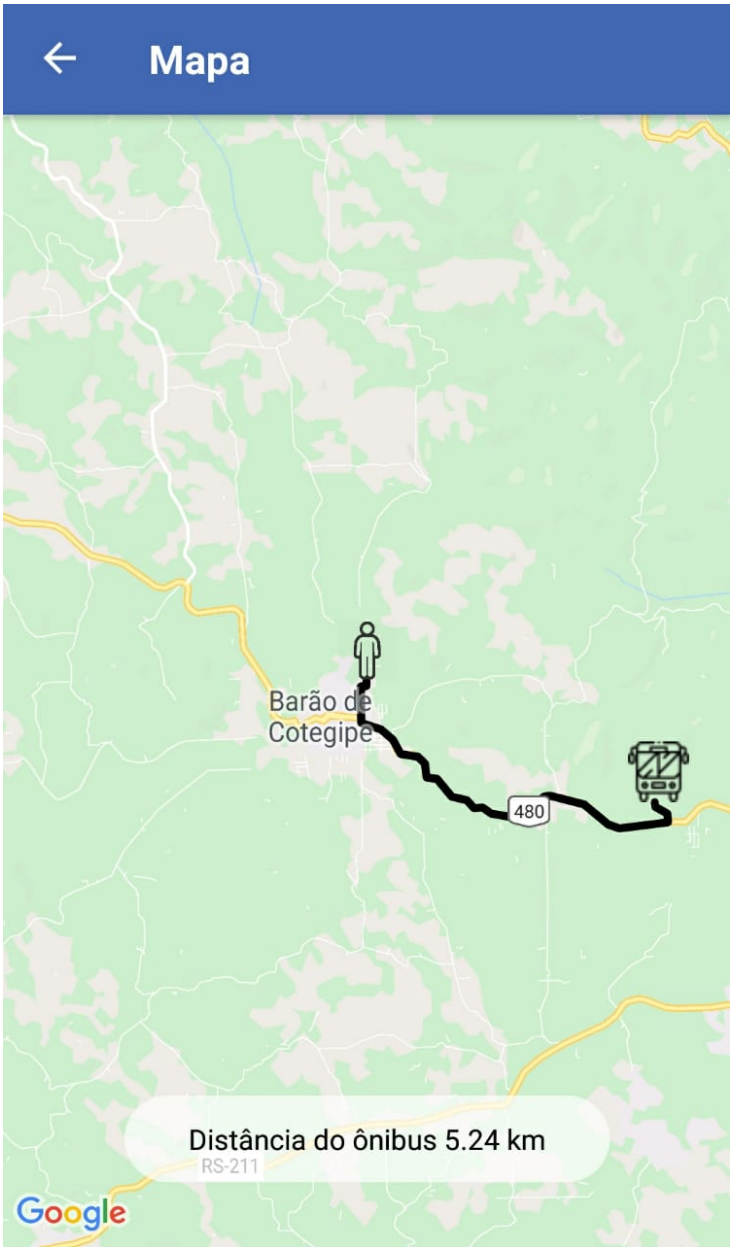
Figura 30 – Função que busca dados sobre veículos.

```
fetch(URL)
  .then(response => response.json())
  .then(responseJson => {
    return responseJson;
  })
  .then(data => {
    setDATA(data);
  })
  .catch(error => {
    console.error(error);
  });
```

Fonte: Autor

Após selecionar um veículo, o mapa onde é mostrado o rastreamento do ônibus, que consiste em mostrar a posição do usuário, posição do ônibus e a distância do usuário em relação ao veículo. Segue Figura 31 mostrando a tela de rastreamento através do mapa.

Figura 31 – Tela que mostra mapa de rastreamento do veículo.



Fonte: Autor

Para realizar a implementação do mapa, foi usada a API do *Google Maps*. Foi necessário criar uma conta na *Google Cloud Platform*. Após possuir a conta criada, é recebido uma chave de acesso e, com ela, é possível utilizar a ferramenta no aplicativo. Este serviço é pago, porém possui um limite grátis que é suficiente para o uso da aplicação em questão.

O *Google Cloud Platform* oferece diversas APIs, porém a que foi mais utilizada no desenvolvimento do projeto foi a *Directions API*, que consiste em mostrar os caminhos entre pontos do mapa.

Para a implementação do projeto *React Native*, primeiramente fora construída uma função que busca os dados das coordenadas na API, através da URL: *https://busfinderapi.herokuapp.com/api/v1/coordinates*. É parametrizada a chave de acesso nos *headers* da requisição. A Figura 32 mostra a função que busca os dados na API.

Figura 32 – Função que busca as coordenadas do ônibus.

```
const getDataBus = () => {
  fetch(URL, {
    method: 'GET',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      Authorization: API_TOKEN,
    },
  })
    .then(response => response.json())
    .then(responseJson => {
      return responseJson;
    })
    .then(data => {
      setLatitude(data[0].latitude);
      setlongitude(data[0].longitude);
    })
    .catch(error => {
      console.error(error);
    });
};
```

Fonte: Autor

Para atualizar a posição do ônibus no mapa foi utilizada a função *setInterval*, do *JavaScript*. A função realiza a atualização a cada segundo, apresentada na Figura 33.

Figura 33 – Função que atualiza posição do ônibus.

```
useEffect(() => {
  setInterval(() => getDataBus(), 1000);
});
```

Fonte: Autor

Para se obter a localização do usuário foi utilizado a biblioteca *Geolocation* do *React Native*. E para calcular a distância entre a posição do ônibus e do usuário foi utilizado a função *haversine* do *JavaScript*. Estas duas implementações estão demonstradas na Figura 34.

Figura 34 – Função que obtém posição do usuário e que calcula distância.

```
useEffect(() => {
   Geolocation.getCurrentPosition(position => {
     setUserLatitude(position.coords.latitude);
     setUserLongitude(position.coords.longitude);
   });
   setDistance(
     haversine(
       {latitude, longitude},
       {latitude: userLatitude, longitude: userLongitude},
       {unit: 'miler'},
     ),
   );
 }, [latitude, longitude, userLatitude, userLongitude]);
```

Fonte: Autor

Com as telas e códigos descritos acima, foi possível elucidar as partes principais do projeto da aplicação móvel, considerando que o mesmo possui muito mais componentes e estruturas que não foram citados.

4 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O objetivo inicial do presente trabalho era construir um aplicativo capaz de monitorar a posição de veículos escolares em tempo real, coletando dados através de um módulo embarcado. Para desenvolver o projeto foi necessário conhecimentos de diversas áreas e tecnologias completamente distintas. Com isso, é possível dizer que o objetivo foi alcançado, descrevendo cada parte do projeto a seguir.

Para desenvolver o servidor através de uma API *Rest*, foi necessário adquirir conhecimentos no *framework Ruby On Rails*, que é implementado através da linguagem *Ruby*. A API foi utilizada como centralizadora dos dados e, por sua vez, armazenando os mesmo no banco relacional *PostgreSQL*.

O módulo de coleta das coordenadas do veículo foi realizada através de um módulo GSM/GPRS, que foi acoplado a um Arduino Uno R3. Para realizar a configuração do módulo embarcado foi necessário escrever um código na linguagem de programação C. Esta parte do processo foi a mais trabalhosa, pois a documentação nessa área é muito escassa e foi necessário testar diversos tipos de configurações, códigos, comandos em *hardware* e recorrer a discussões em fóruns na *web* para conseguir solucionar o problema.

O desenvolvimento do aplicativo móvel foi realizado com a ferramenta *React Native*, onde é possível desenvolver aplicativos nativos para *android* e IOS. O aplicativo foi a outra ponta no processo, comunicando-se com a API, permitindo que o usuário interaja com o sistema.

Como sugestão para trabalhos futuros, é possível melhorar a estrutura do aplicativo, implementando um *chat* onde usuários poderiam interagir entre si. Desenvolver um painel administrativo para motoristas ou gestores de companhias de transportes. Também, disponibilizar dados no perfil do usuário como, por exemplo, históricos de viagens e alertas através de notificações.

Na parte do sistema embarcado algo que pode ser revisto é a questão da alimentação do módulo, pois é necessário uma fonte externa de alimentação de 12V, o que pode acarretar em transtornos na instalação e ocupação de espaço no veículo. Também é considerável pensar em uma estrutura para servir como embalagem do sistema embarcado como um todo.

REFERÊNCIAS

ALMEIDA, M. **Seguro de carro com rastreador é mais barato, mas vale a pena?** 2017. Disponível em: <https://exame.abril.com.br/seu-dinheiro/seguro-de-carro-com-rastreador-e-mais-barato-mas-vale-a-pena/>. Acesso em: 8 de novembro de 2019. Citado na página 20.

CâMARA, R. **O que você deve saber sobre o funcionamento do React Native**. 2018. Disponível em: <https://tableless.com.br/o-que-voce-deve-saber-sobre-funcionamento-react-native/>. Acesso em: 4 de junho de 2019. Citado na página 27.

DEMARTINI, F. **Brasil já tem mais de um smartphone ativo por habitante**. 2018. Disponível em: <https://canaltech.com.br/produtos/brasil-ja-tem-mais-de-um-smartphone-ativo-por-habitante-112294/>. Acesso em: 10 de abril de 2019. Citado na página 26.

ELAINA, J. **Rastreador de carro deixa seguro mais barato**. 2019. Disponível em: <https://www.smartia.com.br/blog/rastreador-carro-seguro-mais-barato/>. Acesso em: 8 de novembro de 2019. Citado na página 20.

FACEBOOK. **React Native**. 2019. Disponível em: <https://facebook.github.io/react-native/>. Acesso em: 1 de novembro de 2019. Citado na página 33.

FUENTES, V. B. **Ruby on Rails: Coloque sua aplicação web nos trilhos**. 1. ed. São Paulo, SP – Brasil: Casa do Código, 2012. Citado 2 vezes nas páginas 32 e 33.

GOOGLE. **Plataforma do Google Maps Documentação**. 2019. Disponível em: <https://developers.google.com/maps/documentation>. Acesso em: 7 de novembro de 2019. Citado 2 vezes nas páginas 34 e 35.

MADUREIRA, D. **APLICATIVO NATIVO, WEB APP OU APLICATIVO HÍBRIDO?** 2017. Disponível em: <https://usemobile.com.br/aplicativo-nativo-web-hibrido/>. Acesso em: 4 de junho de 2019. Citado 2 vezes nas páginas 27 e 28.

MANTUANO, G. **Como funciona o rastreamento veicular?** 2018. Disponível em: <http://blog.softruck.com/como-funciona-o-rastreamento-veicular/>. Acesso em: 10 de maio de 2019. Citado 2 vezes nas páginas 19 e 20.

MCROBERTS, M. **Arduino Básico**. 1. ed. São Paulo, SP – Brasil: Novatec Editora Ltda, 2011. Citado na página 21.

MCWHAE, R. **Ruby on Rails in 2019**. 2019. Disponível em: <https://medium.com/swlh/ruby-on-rails-in-2019-ae3ec463b8bb>. Acesso em: 24 de outubro de 2019. Citado na página 32.

MOTA, A. **O QUE É ARDUINO E COMO FUNCIONA?** 2017. Disponível em: <https://portal.vidadesilicio.com.br/o-que-e-arduino-e-como-funciona/>. Acesso em: 4 de junho de 2019. Citado na página 21.

MOURãO, R. **Mercado de aplicativos e smartphones e as suas estatísticas**. 2017. Disponível em: <http://blog.portalrmfactory.com.br/mercado-de-aplicativos-e-smartphones/>. Acesso em: 1 de junho de 2019. Citado na página 26.

PIROTTI, R. P.; ZUCCOLOTTO, M. Transmissão de dados através de telefonia celular: arquitetura das redes gsm e gprs. **Revista Liberato**, v. 10, n. 13, p. 81–89, 2009. Citado 6 vezes nas páginas 14, 15, 16, 17, 18 e 19.

POSTGRESQL. **What is PostgreSQL?** 2019. Disponível em: <https://www.postgresql.org/docs/9.4/intro-whatis.html>. Acesso em: 15 de novembro de 2019. Citado na página 33.

ROSA, N. **Apps receberam 113 bilhões de downloads na App Store e Google Play em 2018**. 2018. Disponível em: <https://canaltech.com.br/apps/apps-receberam-113-bilhoes-de-downloads-na-app-store-e-google-play-em-2018-129591/>. Acesso em: 1 de junho de 2019. Citado na página 26.

SCORZA, L. **A História do Rails**. 2016. Disponível em: <https://medium.com/codando/a-hist%C3%B3ria-do-rails-87585134f61f>. Acesso em: 23 de outubro de 2019. Citado na página 32.

SCUDERO, E. **A trajetória de um Desenvolvedor Mobile: tudo que você precisa saber!** 2017. Disponível em: <https://becode.com.br/trajetoria-de-um-desenvolvedor-mobile/>. Acesso em: 3 de junho de 2019. Citado 2 vezes nas páginas 26 e 27.

SILVA, C. E. T. d.; ROCHA, C. H. d.; FREIRE, V. S. **Sistemas de Comunicações Pessoais O Padrão GSM**. 2006. Trabalho Final de Graduação, Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro. Citado 3 vezes nas páginas 14, 15 e 17.

SILVA, J. D. T. **Tipos de dados em Ruby**. 2015. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-em-ruby/33600>. Acesso em: 20 de outubro de 2019. Citado na página 31.

SOUZA, L. **Ruby - Aprenda a programar na linguagem mais divertida**. 1. ed. São Paulo, SP – Brasil: Casa do Código, 2014. Citado na página 31.

SUMARES, G. **Número de downloads e receita gerada por aplicativos batem recorde global**. 2017. Disponível em: <https://olhardigital.com.br/pro/noticia/numero-de-downloads-e-receita-gerada-por-aplicativos-batem-recorde-global/71922>. Acesso em: 20 de maio de 2019. Citado na página 26.

SVERZUT, J. U. **Redes GSM, GPRS, EDGE e UMTS: Evolucao a Caminho da Quarta Geracao (4G)**. 4. ed. São Paulo, SP – Brasil: Editora Érica, 2015. Citado 2 vezes nas páginas 18 e 19.

TAVARES, R. **Diferença entre o Arduino Uno, Mega, Nano e Mini**. 2018. Disponível em: <https://www.arduino.blog.br/diferenca-arduino-uno-mega-nano-mini.html>. Acesso em: 15 de setembro de 2019. Citado 3 vezes nas páginas 21, 22 e 23.

THOMSEN, A. **Enviando SMS e Fazendo Chamadas com o Arduino GSM Shield**. 2014. Disponível em: <https://www.filipeflop.com/blog/tutorial-arduino-gsm-shield/>. Acesso em: 15 de outubro de 2019. Citado 2 vezes nas páginas 29 e 30.

THOMSEN, A. **O que é Arduino?** 2014. Disponível em: <https://www.filipeflop.com/blog/o-que-e-arduino/>. Acesso em: 5 de junho de 2019. Citado 3 vezes nas páginas 21, 24 e 25.

THOMSEN, A. **Qual Arduino Comprar? Conheça os Tipos de Arduino**. 2014. Disponível em: <https://www.filipeflop.com/blog/tipos-de-arduino-qual-comprar/>. Acesso em: 25 de setembro de 2019. Citado 2 vezes nas páginas 23 e 24.

ZANDAVALLE, B. B.; SILVA, G. S. D. **SISTEMA DE RELACIONAMENTO PARA PRÁTICA DE ESPORTES**. Florianópolis, Brasil, 2018. Trabalho de Conclusão de Curso, Sistemas De Informação da Universidade do Sul de Santa Catarina. Citado na página 33.