UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JHONATAN MARTINELLO

DESENVOLVIMENTO DE SISTEMA MOBILE PARA O CONTROLE DE VAGAS DE ESTACIONAMENTO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

JHONATAN MARTINELLO

DESENVOLVIMENTO DE SISTEMA MOBILE PARA O CONTROLE DE VAGAS DE ESTACIONAMENTO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. João Guilherme Brasil Pichetti

PATO BRANCO
2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

DESENVOLVIMENTO DE SISTEMA MOBILE PARA O CONTROLE DE VAGAS DE ESTACIONAMENTO

POR

JHONATAN MARTINELLO

Este trabalho de conclusão de curso foi apresentado no dia 05 de dezembro de 2018, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Prof. Esp. João Guilherme Brasil Pichetti
Orientador

Prof. MSc Robison Cris Brito

Prof MSc. Vinicius Pegorini

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

MARTINELLO, Jhonatan. Desenvolvimento de sistema mobile para o controle de vagas de estacionamento. 2018. 54f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2018.

Com a crescente circulação de veículos, fortemente localizada nos centros das cidades, tornou-se quase que uma obrigação aos proprietários de comércios a disponibilização de um local de estacionamento seguro e organizado para seus clientes. Porém como a taxa de circulação de veículos cresceu de forma muito grande e muito rapidamente, a infraestrutura das cidades não acompanhou este crescimento, e consequentemente os comércios locais arcaram com a impossibilidade de um controle de qualidade de seus próprios estacionamentos, devido ao grande fluxo de veículos. Em consequência a esses fatos os motoristas são os mais prejudicados, pois necessitam encontrar uma vaga de forma rápida e segura, porém dificilmente conseguem. Assim, o desenvolvimento de um aplicativo mobile, que esteja integrado a uma rede de sensores nas vagas de estacionamento e que ofereça ao usuário a opção de que ele verifique antes mesmo da chegada ao estabelecimento todas as vagas disponíveis e também as ocupadas trará uma maior facilidade e agilidade aos motoristas na hora de encontrar uma vaga disponível. O aplicativo visa dar ao seu usuário uma melhor forma de encontrar uma vaga para estacionar seu veículo, por meio de um aplicativo o usuário terá uma tela onde constam todas as vagas do estacionamento, e juntamente a vaga a informação de seu *status*, podendo este ser vaga ocupada ou disponível. Este trabalho apresenta a modelagem de um sistema mobile que traz uma forma mais cômoda e ágil para que o motorista encontre uma vaga de estacionamento para seu veículo em um estacionamento privado. Para o desenvolvimento do sistema proposto foram utilizados sensores de proximidade instalados nas vagas para obtenção das distancias, após obtida a distância, a mesma é transmitida ao Raspberry Pi onde é realizada a verificação da distância e conexão com o banco de dados para a atualização do *status* da vaga. Por fim o aplicativo recupera as informações do banco de dados e as apresenta em tela para o usuário.

Palavras-chave: Estacionamento Inteligente. Controle de vagas de estacionamento.

ABSTRACT

MARTINELLO, Jhonatan. Development of mobile system for the control of parking spaces. 2018. 54f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2018.

With the growing circulation of vehicles, heavily located in city centers, it has become almost an obligation for shop owners to provide a secure and organized parking space for their customers. However, as the vehicle circulation rate grew very rapidly, the infrastructure of the cities did not keep up with this growth, and consequently the local trades were unable to control their own parking lots due to the large flow of traffic vehicles. As a result, drivers are the most disadvantaged because they need to find a vacancy quickly and safely, but they can hardly do it. Thus, the development of a mobile app, which is integrated with a sensor network in the parking spaces, and which offers the user the option of verifying the arrival of all available and occupied parking spaces, will bring greater ease and agility for drivers when it comes to finding a vacancy. The app aims to give your user a better way to find a parking space for your vehicle. Through the app the user will have a screen with all the vacancies of the parking, and along the vacancy the information of its status, being able to be vacant occupied or available. This work presents the modeling of a mobile system that brings a more comfortable and agile way for the driver to find a parking space for his vehicle in a private parking. For the development of the proposed system were used proximity sensors installed in the parking spaces to obtain the distances, after obtaining the distance, it is transmitted to the Raspberry Pi where the verification of the distance and connection with the database is carried out to update the status of the parking space. Finally, the app retrieves the information from the database and presents it to the user.

Keywords: Parking Smart. Control of parking spaces.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DOS CÓDIGOS

LISTA DE SIGLAS

HDMI	*High Definition Multimedia Interface*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

LISTA DE ACRÔNIMOS

SINDIPEÇAS Sindicato Nacional da Indústria de Componentes para Veículos Automotores

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Com a crescente necessidade de locomoção de forma barata e rápida, o aumento da frota de automóveis já é uma realidade. Segundo dados do Sindicato Nacional da Indústria de Componentes para Veículos Automotores (SINDIPEÇAS), no Brasil, a frota circulante de automóveis no ano de 2016 foi de aproximadamente 42,9 milhões de unidades. Se levarmos em conta todos os veículos em circulação no país a relação entre a população residente e a frota de auto veículos é de aproximadamente quatro habitantes por veículo (RELATÓRIO..., 2017, p.1).

Diante desta grande quantidade de veículos circulando nos grandes centros já se pode dizer que virou quase uma obrigação de grandes comércios como supermercados e shoppings, por exemplo, disponibilizar aos seus clientes um estacionamento em suas dependências. Porém é de nosso conhecimento que muitas vezes, quando há um grande movimento no local, se torna difícil encontrar uma vaga disponível de forma rápida, este fato acaba muitas vezes prejudicando os clientes que levam muito tempo para encontrar vagas disponíveis.

A automação de estacionamentos proporciona um melhor aproveitamento das vagas existentes, ordenando o fluxo e a circulação de veículos, e fornecendo uma completa informação gerencial para o controle eficaz das vagas.

Diante do exposto, este trabalho visa desenvolver um sistema mobile para automação das vagas de estacionamento de comércios em geral, para que os clientes assim que chegarem ao estacionamento possam verificar locais com vagas disponíveis para estacionar seus automóveis, evitando que os clientes gastem um tempo considerável até encontrar uma vaga disponível. Para o desenvolvimento do projeto foi utilizado o Raspberry Pi, um computador do tamanho de um cartão de crédito que utiliza um sistema operacional baseado no Linux Debian. Através dele é possível desenvolver projetos em diversas linguagens, como phyton por exemplo. Conectados ao Raspberry Pi há sensores de proximidade que indicam se a vaga está ou não disponível. Através da comunicação entre o Raspberry Pi e o aplicativo mobile é possível verificar em tempo real todas as vagas, e se as mesmas estão disponíveis ou ocupadas. Os usuários do sistema podem visualizar todas as informações sobre as vagas

através do aplicativo. O foco principal do sistema é fazer com que os clientes encontrem uma vaga disponível de forma mais fácil e ágil.

1.2 OBJETIVOS

A seguir serão apresentados os objetivos gerais e específicos sobre o presente trabalho.

1.2.1 Objetivo Geral

Desenvolver um sistema mobile para automação das vagas de estacionamento de comércios em geral.

1.2.2 Objetivos Específicos

- Agilizar a localização de vagas no estacionamento.
- Permitir o melhor aproveitamento das vagas de um estacionamento.

1.3 JUSTIFICATIVA

Diante dos visíveis problemas causados pelo aumento significativo do fluxo de veículos nas cidades, e da necessidade de oferecer aos motoristas uma melhora no gerenciamento das vagas de estacionamento de comércios em geral, que em sua maioria não trazem efetividade ao cliente limitado no quesito agilidade, a automatização das vagas de estacionamento de comércios vem ganhando força, e visa proporcionar ao comerciante um maior controle e um melhor aproveitamento das vagas de seu estacionamento, e também visa solucionar o problema do motorista quanto a demora para se encontrar uma vaga disponível.

Este trabalho propõe a automatização das vagas de estacionamento de comércios, a fim de trazer uma melhoria significativa no gerenciamento das mesmas, trazendo ao comerciante um maior controle de suas vagas, e levando maior agilidade e segurança aos clientes que necessitam de uma vaga de estacionamento. Os clientes podem verificar as vagas disponíveis e ocupadas por meio do aplicativo, que apresenta todas as vagas do estacionamento, e também a opção para que o cliente verifique informações sobre o *status*, código e tipo das vagas.

Para o sistema proposto foi escolhido o desenvolvimento de um aplicativo mobile na plataforma Android, tendo em vista que segundo Zuriarrain (2017) o Android passou o Windows e já é o sistema operacional mais usado do mundo, e também pelo fato do crescimento do uso de aplicativos mobile. Para o controle das vagas foi utilizado a plataforma Raspberry Pi, que para Monk (2013, p. 17) “é um computador que executa o sistema operacional Linux, contando com um pacote de edição de texto, capacidades de reprodução de vídeo, jogos e tudo o mais”. Também foram utilizados sensores de proximidade que tem o papel de verificar se a vaga está disponível ou não.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos dos quais este é o primeiro e apresenta as considerações iniciais, objetivos e a justificativa do trabalho. O Capítulo 2 apresenta o referencial teórico. No Capítulo 3 são apresentados os materiais e o método utilizados para o desenvolvimento do trabalho. No Capítulo 4 está o resultado da realização do trabalho, que conta com a modelagem, a apresentação e a implementação do sistema.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a fundamentação teórica deste trabalho, cujo conteúdo explana sobre o crescimento do fluxo de veículos e sobre aspectos técnicos referentes ao Raspberry Pi e Android.

2.1 AUMENTO DO FLUXO DE VEÍCULOS

No o decorrer dos anos foi imenso o crescimento da frota circulante de veículos no Brasil, segundo dados do SINDIPEÇAS a frota circulante de veículos cresceu cerca de 0,7% de 2015 para 2016, chegando a um número total de 42,9 milhões de unidades circulantes (RELATÓRIO..., 2017, p.1).

Na figura 1 é possível verificar o crescimento ano a ano desde 1995 quando o número da frota de veículos era de pouco mais de 16 milhões, até 2016 quando chegou ao número de 42,9 milhões de unidades.

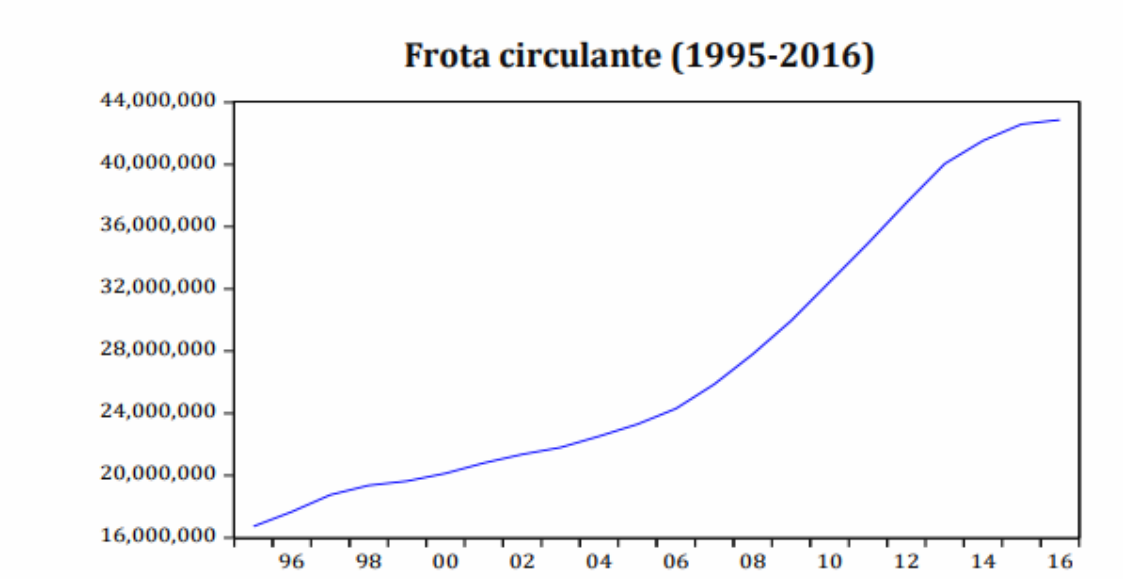

Figura 1: Frota circulante 1995-2016.
Fonte: SINDIPEÇAS

Grande parte deste fluxo de veículos está predominantemente concentrada nos estados de São Paulo, com 36,8% do total, Minas Gerais, com 10,4%, Rio de Janeiro, com 9%, Rio Grande do Sul, com 8,5% e Paraná, com 8,1%. Juntos, esses cinco Estados somam 72,8% de todos os veículos que transitam no País (RELATÓRIO..., 2017, p.1).

Levando em conta todo este crescimento e também o fato de um grande número de veículos estarem localizados em estados e cidades específicas, se torna quase que impossível a infraestrutura das cidades acompanharem todo este crescimento, fato que se torna perceptível verificando os grandes engarrafamentos e a falta de espaços para estacionar nas cidades.

Percebendo todos esses fatores muitos comerciantes já estão disponibilizando estacionamentos privados, temendo perder clientes devido à falta de um local adequado para deixarem seus veículos. Porém devido ao grande fluxo de veículos entrando e saindo dos estacionamentos é muito comum que os próprios comerciantes não tenham total controle de suas vagas de estacionamento, e também que clientes acabem perdendo muito tempo até encontrar uma vaga disponível, fazendo até com que muitos desistam de ir ao estabelecimento.

A automação de vagas de estacionamento vem como a solução para o problema nos estacionamentos privados, pois automatizando o estacionamento o comerciante terá um maior controle de suas vagas e o cliente por sua vez gastará muito menos tempo para encontrar uma vaga disponível.

2.2 TECNOLOGIAS, PLATAFORMAS E FERRAMENTAS

2.2.1 Arduino

Na etapa inicial do trabalho, estava previsto a utilização da plataforma Arduino para a função de gerenciamento dos sensores de proximidade e para a realização da conexão junto ao Firebase. Inicialmente na etapa de gerenciamento dos sensores o Arduino se saiu muito bem, realizando com êxito a função destinada. Porém na etapa de conexão com o Firebase, devido ao fato da plataforma necessitar de um módulo WiFi externo, a complexidade encontrada foi muito grande. Foram utilizados o módulo ESP8266 que não obteve êxito, nem ao menos foi possível conectar-se à rede através deste módulo, e o módulo ESP8266 NodeMCU o qual conectava-se à rede, porém por se tratar de um micro controlador o mesmo se tornou muito complexo para o desenvolvimento do projeto.

Após o revés na utilização do Arduino, chegou-se à conclusão de que maiores êxitos seriam alcançados com a utilização da plataforma Raspberry Pi, para essa função destinada ao Arduino.

2.2.2 Raspberry Pi

Para Monk (2013, p. 17) o Raspberry Pi “é um computador que executa o sistema operacional Linux. Ele tem portas USB nas quais você pode conectar um teclado e um mouse e uma saída de vídeo *High Definition Multimedia Interface* (HDMI) na qual você pode conectar uma TV ou um monitor”. Ainda segundo Monk (2013, p. 17) o Raspberry Pi “é realmente um computador completo, contando com um pacote de edição de texto, capacidades de reprodução de vídeo, jogos e tudo o mais”.

Após o revés na utilização da plataforma Arduino, foi escolhido para o desenvolvimento do projeto o Raspberry Pi pelo fato de ser uma plataforma de baixo custo, que dispõe de todas as funcionalidades para alcançar o êxito ao fim do projeto. A placa dispõe de meios para que seja possível receber dados de sensores, e para que a comunicação com o aplicativo mobile seja possível.

2.2.3 Sensores de proximidade

O Raspberry Pi oferece diversas opções de sensores que podem ser conectados a ele, tais como, sensores de temperatura, pressão e proximidade, o qual foi o escolhido para o desenvolvimento do projeto, mais precisamente o sensor ultrassônico HC-SR04. Mais informações sobre o sensor estão disponíveis no capitulo 3 Materiais e Método do trabalho.

2.2.4 Comunicação entre Raspberry Pi e aplicativo mobile

Para a comunicação entre o Raspberry Pi e o aplicativo mobile foi utilizado a ferramenta Firebase, que oferece dentre outras funcionalidades um banco de dados NoSQL, que utiliza *JavaScript Object Notation* (JSON) para armazenamento de dados em nuvem. Mais informações sobre o Firebase estão disponíveis no capitulo 3 Materiais e Método do trabalho.

As informações sobre as vagas são capturadas pelos sensores, transmitidas ao Raspberry Pi e gravadas no banco de dados. O aplicativo busca as informações do banco de dados e mostra em tela para o usuário.

2.2.5 Android

Segundo da Silva (2015, p. 4) “o Android é uma plataforma aberta voltada para dispositivos móveis desenvolvida pela Google e atualmente é mantida pela *Open Handset Alliance* (OHA)”. Foi lançado em 2007 pela Google e tem seu código do sistema operacional disponibilizado pelo Google sob licença de código aberto.

Atualmente, de acordo com Zuriarrain (2017) o Android passou o Windows e já é o sistema operacional mais usado do mundo.

Desde abril de 2009 as versões do Android têm sido batizadas de codinomes de doces e respeitando uma ordem alfabética. Na figura 2 é possível ver todas as versões e suas porcentagens de distribuição a partir da 2.3.3 Gingerbread até a mais atual 8.0 Oreo. Para o desenvolvimento do projeto foi utilizada a versão 6.0 Marshmallow que é a mais utilizada atualmente (novembro 2017).

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.6%
4.1.x	Jelly Bean	16	2.3%
4.2.x		17	3.3%
4.3		18	1.0%
4.4	KitKat	19	14.5%
5.0	Lollipop	21	6.7%
5.1		22	21.0%
6.0	Marshmallow	23	32.0%
7.0	Nougat	24	15.8%
7.1		25	2.0%
8.0	Oreo	26	0.2%

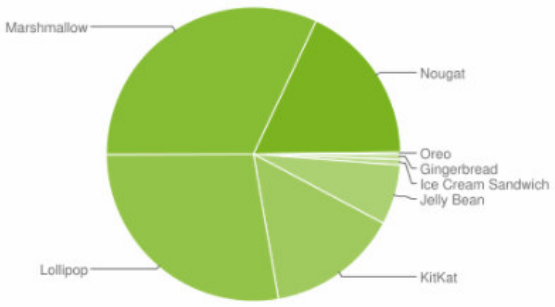

Figura 2: Versões Android.
Fonte: ANDROID DEVELOPERS

3 MATERIAIS E MÉTODO

Neste capítulo, serão apresentados os materiais e os métodos utilizados para alcançar o objetivo deste trabalho.

3.1 MATERIAIS

O Quadro 1 apresenta a lista de ferramentas e tecnologias que foram utilizadas para o desenvolvimento deste trabalho.

Ferramenta / Tecnologia	Versão	Finalidade
Visual Paradigm Community Edition	14.1	Modelagem UML (casos de uso e diagrama de classes).
Firebase		Ferramenta de gerenciamento do lado servidor.
Raspberry Pi	Model B	Gerenciar sensores e fazer a conexão junto ao Firebase.
Android Studio	3.1.4	*Integrated Development Environment* (IDE) para desenvolvimento do aplicativo Android.
Java	8	Linguagem de Programação
Python	3.5	Linguagem de Programação
Sensor ultrassônico	HC-SR04	Identificar movimentação de veículos na vaga.

Quadro 1: Lista de ferramentas e tecnologias

3.1.1 Visual Paradigm Community Edition

"Visual Paradigm é uma ferramenta de software projetada para equipes de desenvolvimento de software para modelar sistema de informações empresariais e gerenciar processos de desenvolvimento" (VISUAL PARADIGM, 2017). É uma ferramenta gratuita voltada para a modelagem de diagramas *Unified Modeling Language* (UML).

A ferramenta Visual Paradigm Community Edition segundo VISUAL PARADINGM (2017) oferece suporte para os 13 tipos de diagramas UML oferecidos, são eles: diagrama de classe, caso de uso, sequência, comunicação, máquina de estado, atividade, componentes, implantação, pacote, objeto, estrutura composta, perfil, tempo e visão geral da interação.

3.1.2 Google Firebase

O Firebase "oferece suporte ao compartilhamento de recursos, como banco de dados, configurações e notificações entre apps multiplataformas" (FIREBASE, 2017). O Firebase foi desenvolvido com o intuito de facilitar o desenvolvimento de apps (Android e iOS) e de

aplicações web. Ele disponibiliza todos os recursos de *back-end* necessários para o desenvolvedor em um único lugar.

A ferramenta também oferece diversas funcionalidades, como, banco de dados NoSQL utilizando JSON para armazenamento de dados, suporte para autenticação de usuários via e-mail, Facebook, GitHub, Google Sign-In e Twitter, relatórios de erros, e *insights* sobre o app.

3.1.3 Raspberry Pi

O Raspberry Pi é um computador pequeno, portátil e barato, que se conectado a um monitor de computador ou TV, e a um teclado e mouse padrão, pode ser utilizado para realizar diversas tarefas, como desenvolvimento de códigos de programação em diversas linguagens. O Raspberry Pi foi desenvolvido no Reino Unido pela Fundação Raspberry Pi.

De acordo com RASPBERRY PI (2018) a plataforma utiliza o Linux Debian como sistema operacional oficial para todos os modelos do Raspberry Pi.

3.1.4 Android Studio

O Android Studio foi lançado pelo Google para o desenvolvimento de aplicativos para o sistema operacional Android. Oferece um editor de código inteligente, que ajuda a trabalhar mais rápido e aumentar a produtividade oferecendo preenchimento automático de código avançado, refatoração e análise de código.

A *Integrated Development Environment* (IDE) possui um emulador para testes dos aplicativos, o qual simula um dispositivo real e facilita a realização de testes e a execução do aplicativo desenvolvido. Outra funcionalidade da IDE é a opção de *preview* de *layout,* onde é possível verificar como está ficando todo o *layout* da tela desenvolvida.

A versão do Android escolhida para o desenvolvimento do projeto é a 6.0 Marshmallow, pois está versão dispõe de todas as funcionalidades necessárias para o desenvolvimento da aplicação, e segundo (ANDROID DEVELOPERS, 2017) a versão 6.0 é a mais utilizada no mundo atualmente, correspondendo a 32% dos aparelhos no mercado.

3.1.5 Linguagem de Programação Java

Java é uma linguagem de programação interpretada orientada a objetos. Um ponto fortíssimo do Java é a quantidade enorme de bibliotecas gratuitas para realizar os mais

diversos trabalhos, tais como relatórios, gráficos, sistemas de busca, geração de código de barra, manipulação de *eXtensible Markup Language* (XML), tocadores de vídeo, manipuladores de texto, persistência transparente, impressão, entre outros.

A linguagem de programação Java é utilizada para desenvolvimentos de aplicação desktop, web e móveis.

3.1.6 Linguagem de Programação Python

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

O Python é uma linguagem de programação que permite trabalhar mais rapidamente e integrar seus sistemas com mais eficiência (PYTHON, 2018).

3.1.7 Sensor ultrassônico HC-SR04

O Sensor ultrassônico HC-SR04, utilizado para o desenvolvimento do projeto possui função de medição sem contato de 2 centímetros até 4 metros, com precisão de aproximadamente 3 milímetros.

Conforme pode-se verificar na Figura 3 abaixo, o sensor funciona através do envio de sinais ultrassônicos pelo pino Trigger, esses sinais ao colidirem com o objeto soam um sinal de retorno, que é recebido pelo pino Echo do sensor. Aplicando esse sinal de retorno recebido à uma fórmula matemática, é possível identificar a distância do objeto para com o sensor.

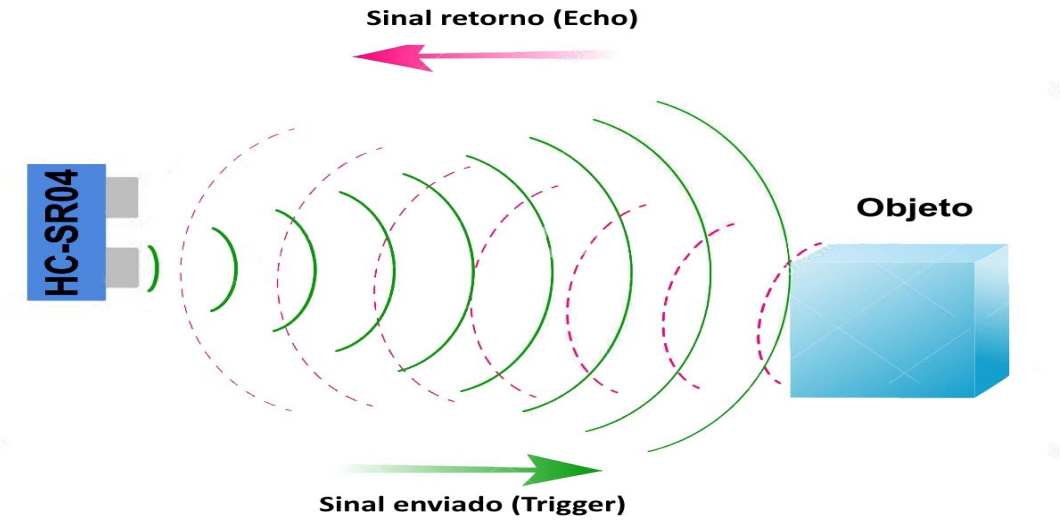

Figura 3: Funcionamento sensor HC-SR04
Fonte: FILIPEFLOP

3.2 MÉTODO

O método escolhido para o desenvolvimento deste trabalho foi o processo sequencial linear de Pressman e Maxim (2016). Nesse modelo as tarefas ocorrem sequencialmente, iniciando pela especificação dos requisitos e seguindo pelas fases de modelagem, construção e disponibilização do sistema, tendo por fim o suporte contínuo do sistema (PRESSMAN, MAXIM, 2016, p. 42). Esse modelo foi utilizado pelo fato de o sistema envolver poucos e bem definidos requisitos, o que faz do modelo sequencial mais vantajoso.

As atividades definidas para o desenvolvimento do trabalho estão descritas a seguir:

a) Análise – Os requisitos foram definidos através da pesquisa e estudo de sistemas consistentes e semelhante ao proposto neste trabalho, e também através do estudo de bibliografias de autores conhecidos na área de Engenharia de Software. Após o levantamento dos requisitos, foi realizada a análise dos mesmos e definido sua estrutura e as formas como irá interagir com o usuário.

b) Projeto – Na etapa de projeto do sistema foi realizada a modelagem e a análise do projeto. Iniciando-se pela construção dos diagramas de caso de uso e os das principais atividades do sistema e posteriormente desenvolvendo a modelagem do banco de dados.

c) Implementação – A implementação do sistema foi realizada utilizando as ferramentas Android Studio e Raspberry Pi. Para controle gerencial dos dados, como operações de inclusão, edição e exclusão no sistema, e para interação do usuário com o

sistema, com acesso a lista de estabelecimentos e suas respectivas vagas, foram desenvolvidos dois aplicativos Android, ambos através do Android Studio. Já para o controle dos sensores de proximidade e para a conexão junto ao Firebase foi utilizado o Raspberry Pi e a linguagem de programação Python. Foram implantados todos os cadastros do sistema, estabelecimentos, tipos vaga e vagas.

d) Testes – Os testes foram informais e realizados à medida que as iterações ocorriam. Esses testes incluíram verificação do código e a forma de interação com o aplicativo.

4 RESULTADOS

Este capítulo apresenta o resultado deste trabalho que é o desenvolvimento de um sistema mobile para controle de vagas de estacionamento.

4.1 ESCOPO DO SISTEMA

O sistema mobile para controle de vagas de estacionamento visa proporcionar a donos de estabelecimentos que disponibilizam estacionamento a seus clientes um melhor aproveitamento das vagas existentes, fazendo com que por meio dele o fluxo e a circulação de veículos fique de forma mais ordenada, fornecendo uma completa informação gerencial para o controle eficaz das vagas. Além de trazer uma melhor experiência de uso do estacionamento aos clientes, assim o sistema terá acesso a todas as informações sobre as vagas existentes e o respectivo *status* de cada vaga do estacionamento, possibilitando assim maior agilidade para encontrar uma vaga disponível.

Para o controle das vagas foram utilizados sensores de proximidade, que tem o papel de verificar se a vaga está disponível ou ocupada. Os dados interceptados pelos sensores são enviados a plataforma Raspberry Pi que verifica e transmite através da conexão via Wi-Fi para a nuvem, disponibilizando a informação em tempo real para o aplicativo mobile, que por sua vez mostra ao usuário instantaneamente todas as vagas do estabelecimento e o respectivo *status* de cada vaga.

Na Figura 4 abaixo pode-se verificar todo o processo de funcionamento do sistema. Inicia-se pela instalação do sensor de proximidade na vaga, o sensor faz a obtenção da distância e envia a mesma para o Raspberry Pi. No Raspberry Pi a distância obtida é verificada afim de saber o novo *status* da vaga, e depois de verificada é feita a conexão junto ao Firebase para a gravação do novo *status*. No Firebase todos os dados relativos às vagas são salvos, dados estes que são usufruídos pelo aplicativo mobile e apresentados em tela ao usuário.

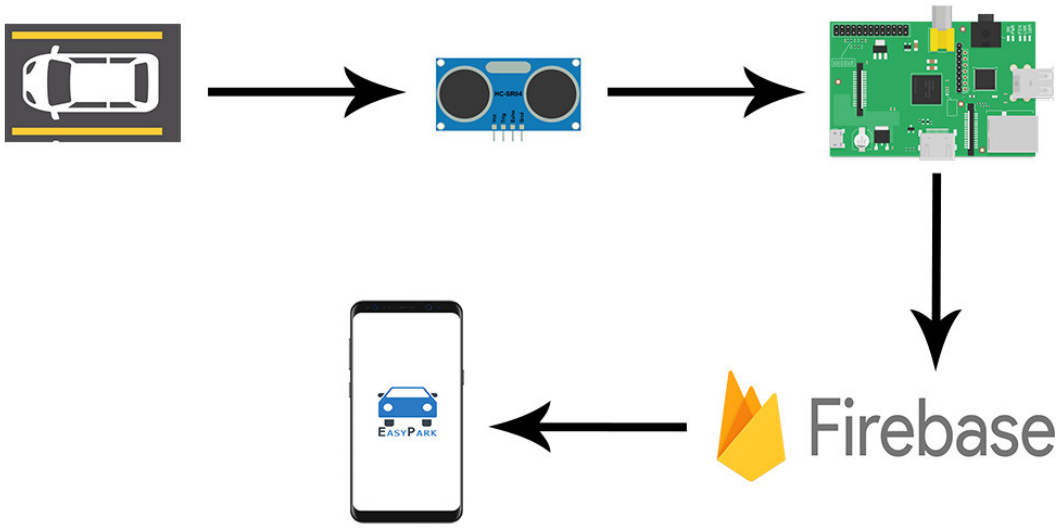

Figura 4: Processo de funcionamento do sistema

O usuário baixará o aplicativo em seu celular, e sempre que estiver dentro da área de alcance de uma rede Wi-Fi, ou utilizando rede celular, poderá se conectar e por meio do aplicativo verificará todas as vagas existentes no estabelecimento desejado e seus respectivos *status*, que poderão ser disponível ou ocupada. O aplicativo também tem o papel de mostrar aos usuários vagas que por ventura sejam exclusivas para idosos ou portadores de necessidades especiais. Para a utilização do aplicativo não é necessário que o usuário faça o seu cadastro, tendo em vista que o aplicativo tem o único papel de mostrar aos usuários todas as vagas de um estabelecimento e seus respectivos *status*.

4.2 MODELAGEM DO SISTEMA

Os Quadros de 2 a 7 apresentam a descrição dos requisitos funcionais e os requisitos não funcionais do sistema. Nos quadros, as siglas RF e RNF são utilizadas para descrever os requisitos funcionais e requisitos não funcionais, respectivamente.

RF1 – Manter Estabelecimentos	
Descrição: O cadastro de estabelecimentos apresentará o nome do estabelecimento, CNPJ, e-mail, senha, rua, número, bairro e cidade.	
Requisitos Não-Funcionais	
Nome	Restrição

RNF 1.1 – Validação	O sistema fará a validação dos campos nome, CNPJ, e-mail e senha.
RNF 1.2 – Validação CNPJ	O sistema não permitirá o cadastro com o campo CNPJ em branco.
RNF 1.3 – Senha do Estabelecimento	A senha do estabelecimento deve conter um mínimo de 6 caracteres.
RNF 1.4 – Alterar	O administrador do sistema poderá alterar os dados do estabelecimento.
RNF 1.5 – Excluir	O administrador do sistema poderá excluir estabelecimentos.

Quadro 2: Requisito manter estabelecimentos.

RF2 – Manter Tipo de Vaga	
Descrição: O cadastro de tipo de vaga apresentará o nome do tipo da vaga.	
Requisitos Não-Funcionais	
Nome	Restrição
RNF 2.1 – Validação	O sistema não permitirá o cadastro de tipos de vaga com o campo nome em branco.
RNF 2.2 – Alterar	O funcionário do estabelecimento poderá alterar os dados dos tipos de vagas.
RNF 2.3 – Excluir	O funcionário do estabelecimento poderá excluir os tipos de vagas.

Quadro 3: Requisito manter tipo de vaga.

RF3 – Manter Vagas	
Descrição: O cadastro de vagas apresentará um código identificador, o *status*, o tipo da vaga e o estabelecimento a que a vaga pertence.	
Requisitos Não-Funcionais	
Nome	Restrição
RNF 3.1 – Validação código identificador	O sistema não permitirá o cadastro de uma vaga com o campo código em branco.
RNF 3.2 – Validação de *status*	O *status* da vaga poderá ser disponível ou ocupada.
RNF 3.3 – Alterar	O funcionário do estabelecimento poderá alterar os dados das vagas.
RNF 2.3 – Excluir	O funcionário do estabelecimento poderá excluir vagas.

Quadro 4: Requisito manter vagas.

RF4 – Visualizar Vagas
Descrição: O usuário, o administrador do sistema e o funcionário do estabelecimento poderão visualizar as vagas e seus respectivos *status*.

Quadro 5: Requisito visualizar vagas.

RF5 – Alterar *status* da vaga	
Descrição: Após o sistema verificar se existe ou não um veículo na vaga, o mesmo deve atualizar o *status* dessa vaga na nuvem.	
Requisitos Não-Funcionais	
Nome	Restrição
RNF 6.1 – Se houver veículo na vaga	Se os sensores identificarem um veículo na vaga o sistema altera o *status* para ocupada.
RNF 6.1 – Se não houver veículo na vaga	Se os sensores identificarem que não há nenhum veículo na vaga o sistema altera o *status* para disponível.

Quadro 6: Requisito alterar status da vaga.

A Figura 5 apresenta o diagrama de casos de uso do sistema. Esse diagrama demonstra as funcionalidades essenciais do sistema que são realizadas por seus atores representados pelo usuário, funcionário do estabelecimento, administrador do sistema e sistema. O usuário poderá visualizar as vagas. O funcionário fará o cadastro dos tipos de vaga e das vagas e também poderá gerenciar os dados do estabelecimento. O administrador do sistema terá a função de efetuar os cadastros de estabelecimentos, enquanto o sistema identificará se existem veículos nas vagas e fará o controle e a alteração do *status* da vaga.

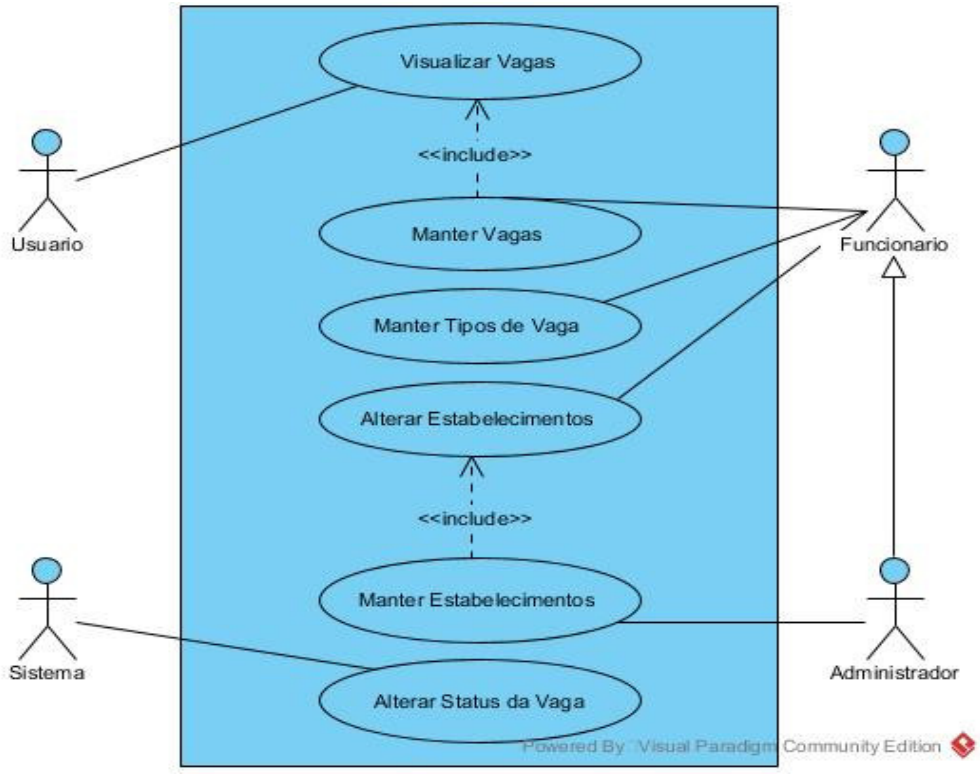

Figura 5: Casos de Uso.

Os Quadros numerados de 8 a 19 apresentam a descrição das operações inserir, consultar, alterar e excluir dos casos de uso apresentados na Figura 5.

Caso de uso:
Inserir estabelecimento (refere-se à operação de inclusão no caso de uso "Manter Estabelecimento").
Descrição:
Ator inclui dados no sistema.
Atores:
Administrador do Sistema.
Pré-condição:
Não há.
Sequência de Eventos:
1. Ator acessa a tela de estabelecimentos, pelo menu inicial.
2. O Ator insere os dados necessários para determinado cadastro.
3. O Ator pressiona o botão Salvar.
4. O sistema insere as informações no banco de dados.

Pós-Condição:
Estabelecimento é inserido no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão "Salvar". 1.2 O sistema valida as informações e exibe mensagem informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de inclusão, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.
2. Campos preenchidos com dados inválidos	2.1. O ator preenche os campos de forma incorreta e clica no botão "Salvar". 2.2. O sistema valida as informações e exibe uma mensagem informando que os dados foram preenchidos de forma incorreta. 2.3. O sistema retorna à tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido

Quadro 7: Operação inserir do caso de uso manter estabelecimentos.

Caso de uso:
Alterar estabelecimento (refere-se à operação de alteração no caso de uso "Manter Estabelecimento").
Descrição:
Ator altera dados do sistema.
Atores:
Administrador do Sistema ou Funcionário do Estabelecimento.
Pré-condição:
Ter um ou mais estabelecimentos cadastrados.
Sequência de Eventos:

1. Ator seleciona qual o cadastro que necessita fazer a alteração. **2.** O Ator altera os dados necessários. **3.** O Ator pressiona o botão Salvar. **4.** O sistema altera as informações no banco de dados. **Pós-Condição:** Informações do estabelecimento são alteradas no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão "Salvar". 1.2 O sistema valida as informações e exibe mensagem informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de alteração, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.
2. Campos preenchidos com dados inválidos	2.1. O ator preenche os campos de forma incorreta e clica no botão "Salvar". 2.2. O sistema valida as informações e exibe uma mensagem informando que os dados foram preenchidos de forma incorreta. 2.3. O sistema retorna à tela de alteração, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido

Quadro 8: Operação alterar do caso de uso manter estabelecimentos.

Caso de uso: Excluir estabelecimento (refere-se à operação de exclusão no caso de uso "Manter Estabelecimento"). **Descrição:** Ator exclui dados do sistema. **Atores:** Administrador do Sistema. **Pré-condição:** Ter um ou mais estabelecimentos cadastrados. **Sequência de Eventos:** **1.** Ator seleciona qual o cadastro que quer excluir. **2.** O Ator pressiona o botão Excluir. **3.** O Ator confirma a exclusão do estabelecimento. **4.** O sistema exclui as informações do banco de dados. **Pós-Condição:** Estabelecimento é excluído do banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Exclusão de registro que possui vínculos no sistema.	1.1. Ator solicita a exclusão do registro que possui vínculos no sistema. 1.2 O sistema verifica que o registro possui vínculos e exibe uma mensagem informando que o registro não pode ser excluído.

Quadro 9: Operação excluir do caso de uso manter estabelecimentos.

Caso de uso:
Inserir tipo de vaga (refere-se à operação de inclusão no caso de uso "Manter Tipos de Vaga").
Descrição:
Ator inclui dados no sistema.
Atores:
Funcionário do Estabelecimento.
Pré-condição:
Ator deve estar logado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para cadastrar os tipos de vaga, pelo menu inicial.
2. O Ator insere os dados necessários para determinado cadastro.
3. O Ator pressiona o botão Salvar.
4. O sistema insere as informações no banco de dados.

Pós-Condição:
Tipo de Vaga é inserido no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão "Salvar". 1.2 O sistema valida as informações e exibe mensagem informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de inclusão, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.

Quadro 10: Operação inserir do caso de uso manter tipos de vaga.

Caso de uso:
Alterar tipo de vaga (refere-se à operação de alteração no caso de uso "Manter Tipos de Vaga").
Descrição:
Ator altera dados do sistema.
Atores:
Funcionário do Estabelecimento.
Pré-condição:
Ator deve estar logado no sistema. Ter um ou mais tipos de vaga cadastrados no sistema.
Sequência de Eventos:
1. Ator seleciona qual o cadastro que necessita fazer a alteração.
2. O Ator altera os dados necessários.
3. O Ator pressiona o botão Salvar.
4. O sistema altera as informações no banco de dados.

Pós-Condição:
Informações do tipo de vaga são alteradas no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão "Salvar". 1.2 O sistema valida as informações e exibe mensagem

	informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de alteração, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.

Quadro 11: Operação alterar do caso de uso manter tipos de vaga.

Caso de uso: Excluir Tipo de Vaga (refere-se à operação de exclusão no caso de uso "Manter Tipos de Vaga"). **Descrição:** Ator exclui dados do sistema. **Atores:** Funcionário do Estabelecimento. **Pré-condição:** Ator deve estar logado no sistema. Ter um ou mais tipos de vaga cadastrados no sistema. **Sequência de Eventos:** **1.** Ator seleciona qual o cadastro que quer excluir. **2.** O Ator pressiona o botão Excluir. **3.** O Ator confirma a exclusão do tipo de vaga. **4.** O sistema exclui as informações do banco de dados. **Pós-Condição:** Tipo de vaga é excluído do banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Exclusão de registro que possui vínculos no sistema.	1.1. Ator solicita a exclusão do registro que possui vínculos no sistema. 1.2 O sistema verifica que o registro possui vínculos e exibe uma mensagem informando que o registro não pode ser excluído.

Quadro 12: Operação excluir do caso de uso manter tipos de vaga.

Caso de uso: Inserir vaga (refere-se à operação de inclusão no caso de uso "Manter Vaga"). **Descrição:** Ator inclui dados no sistema. **Atores:** Funcionário do Estabelecimento. **Pré-condição:** Ator deve estar logado no sistema. Ter um ou mais tipos de vaga cadastrados no sistema. **Sequência de Eventos:** **1.** Ator acessa a tela para cadastro de vagas, pelo menu inicial. **2.** O Ator insere os dados necessários para determinado cadastro. **3.** O Ator pressiona o botão Salvar. **4.** O sistema insere as informações no banco de dados. **Pós-Condição:** Vaga é inserido no banco de dados.	
Nome do fluxo alternativo	**Descrição**

(extensão)	
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão “Salvar”. 1.2 O sistema valida as informações e exibe mensagem informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de inclusão, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.
2. Campos preenchidos com dados inválidos	2.1. O ator preenche os campos de forma incorreta e clica no botão “Salvar”. 2.2. O sistema valida as informações e exibe uma mensagem informando que os dados foram preenchidos de forma incorreta. 2.3. O sistema retorna à tela de inclusão, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido

Quadro 13: Operação inserir do caso de uso manter vaga.

Caso de uso:
Alterar vaga (refere-se à operação de alteração no caso de uso “Manter Vaga”).
Descrição:
Ator altera dados do sistema.
Atores:
Funcionário do Estabelecimento.
Pré-condição:
Ator deve estar logado no sistema. Ter um ou mais tipos de vaga e vagas cadastrados no sistema.
Sequência de Eventos:
1. Ator seleciona qual o cadastro que necessita fazer a alteração.
2. O Ator altera os dados necessários.
3. O Ator pressiona o botão Salvar.
4. O sistema altera as informações no banco de dados.

Pós-Condição:
Informações da vaga são alteradas no banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche um ou mais campos obrigatórios e clica no botão “Salvar”. 1.2 O sistema valida as informações e exibe mensagem informando que campos obrigatórios não foram preenchidos. 1.3. O sistema retorna à tela de alteração, com os campos que haviam sido preenchidos e destacando os campos obrigatórios sem preenchimento.
2. Campos preenchidos com dados inválidos	2.1. O ator preenche os campos de forma incorreta e clica no botão “Salvar”. 2.2. O sistema valida as informações e exibe uma mensagem informando que os dados foram preenchidos de forma incorreta.

	2.3. O sistema retorna à tela de alteração, com os campos que já haviam sido preenchidos e destacando os campos com formato inválido

Quadro 14: Operação alterar do caso de uso manter vaga.

Caso de uso: Excluir Vaga (refere-se à operação de exclusão no caso de uso "Manter Vaga"). **Descrição:** Ator exclui dados do sistema. **Atores:** Funcionário do Estabelecimento. **Pré-condição:** Ator deve estar logado no sistema. Ter um ou mais tipos de vaga e vaga cadastrados no sistema. **Sequência de Eventos:** **3.** Ator seleciona qual o cadastro que quer excluir. **4.** O Ator pressiona o botão Excluir. **5.** O Ator confirma a exclusão da vaga. **6.** O sistema exclui as informações do banco de dados. **Pós-Condição:** Vaga é excluída do banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
7. Exclusão de registro que possui vínculos no sistema.	1.1. Ator solicita a exclusão do registro que possui vínculos no sistema. 1.2 O sistema verifica que o registro possui vínculos e exibe uma mensagem informando que o registro não pode ser excluído.

Quadro 15: Operação excluir do caso de uso manter vaga.

Caso de uso: Visualizar vagas. **Descrição:** Ator visualiza informações das vagas no sistema. **Atores:** Funcionário do Estabelecimento e Usuário. **Pré-condição:** Ter uma ou mais vagas cadastradas no sistema. **Sequência de Eventos:** **5.** O Ator seleciona a opção de visualização das vagas no sistema, pelo menu inicial. **6.** O sistema retorna o mapa de vagas com seus respectivos dados, como, código, tipo de vaga e *status*. **Pós-Condição:** Ator visualiza todo o mapa de vagas.

Quadro 16: Caso de uso visualizar vagas.

Caso de uso: Alterar *status* das vagas. **Descrição:** Sistema altera *status* da vaga.

Atores: Sistema. **Pré-condição:** Ter vagas cadastradas. **Sequência de Eventos:** **1.** O sistema recebe informações do sensor sobre a vaga. **2.** O sistema altera o *status* da vaga de acordo com os dados recebidos. **Pós-Condição:** Sistema altera o *status* da vaga de acordo com os dados recebidos pelo sensor.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Veículo identificado na vaga.	1.1. O sistema verifica que há um veículo estacionado na vaga. 1.2. O sistema altera o *status* da vaga para ocupada.
2. Vaga vazia identificada.	2.1. O sistema verifica que não há nenhum veículo estacionado na vaga. 2.2. O sistema altera o *status* da vaga para disponível.

Quadro 17: Caso de uso alterar status das vagas.

Na Figura 6 é apresentado o diagrama de classes do sistema. O diagrama de classes é composto pela classe Estabelecimento, composta pelos atributos nome, CNPJ, email, senha, endereço, bairro, cidade e estado. Relacionada à classe Estabelecimentos há a classe Vagas, composta dos atributos codigoVaga, tipoVaga, vinda do relacionamento com a classe TipoVaga, estabelecimento, vinda do relacionamento coma classe Estabelecimento e *status*. Também relacionada à classe Vaga há a classe TipoVaga, que é composta unicamente pelo atributo nome.

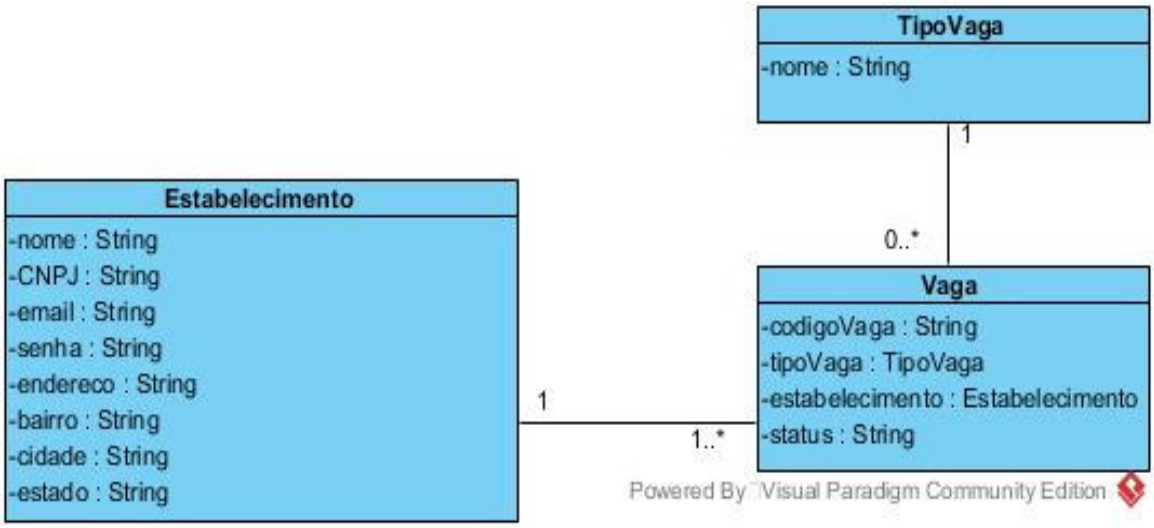

Figura 6: Diagrama de Classes

4.3 APRESENTAÇÃO DO SISTEMA

Esta seção apresenta as funcionalidades e o uso de recursos tecnológicos do sistema por meio de suas telas, enfatizando a interação com o sistema. A apresentação do sistema é feita sob a forma de texto. As telas são tratadas como cópias (*print screen*) das telas do sistema e demonstradas por meio de figuras.

O projeto está dividido entre dois aplicativos mobile, o EasyPark Gerencial e o EasyPark. O EasyPark Gerencial é o aplicativo por onde o estabelecimento fará o gerenciamento das vagas e de seus tipos, e terá o controle de seus dados cadastrais, podendo alterá-los conforme necessário. O EasyPark é o aplicativo no qual o cliente verificará todos os estabelecimentos cadastrados no sistema e todas suas respectivas vagas e *status*. A seguir serão apresentadas todas as informações relativas aos dois aplicativos.

4.3.1 EasyPark Gerencial

A Figura 7 é a representação da tela de autenticação do aplicativo. Esta tela contém os campos e-mail e senha, as quais fazem a autenticação do cadastro da empresa no sistema.

Figura 7: Tela de login EasyPark Gerencial

A Figura 8 é a representação da tela inicial do aplicativo. A tela inicial do aplicativo apresenta uma mensagem de boas vindas ao usuário, e dispõe de um menu lateral no qual se encontram todas as funcionalidades do aplicativo.

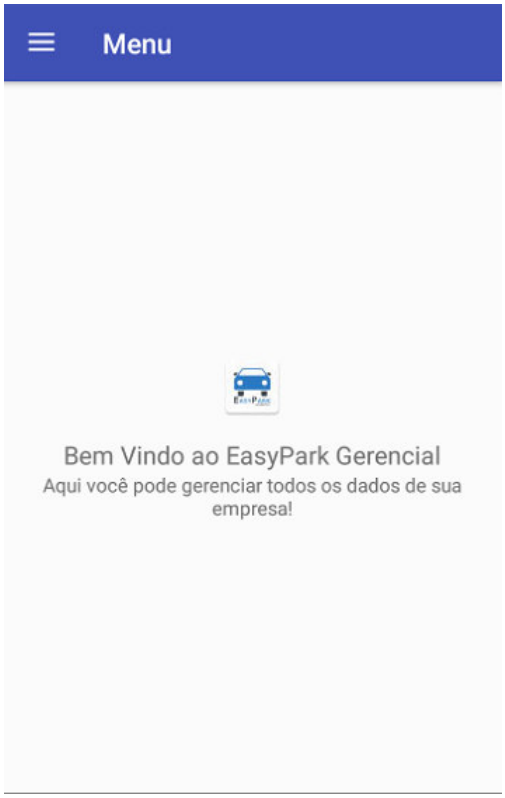

Figura 8: Tela inicial EasyPark Gerencial

A Figura 9 demonstra o menu do aplicativo. O menu contém as opções de gerenciamento de tipos de vaga e gerenciamento de vagas, além das opções de verificação dos dados do estabelecimento e alteração de senha.

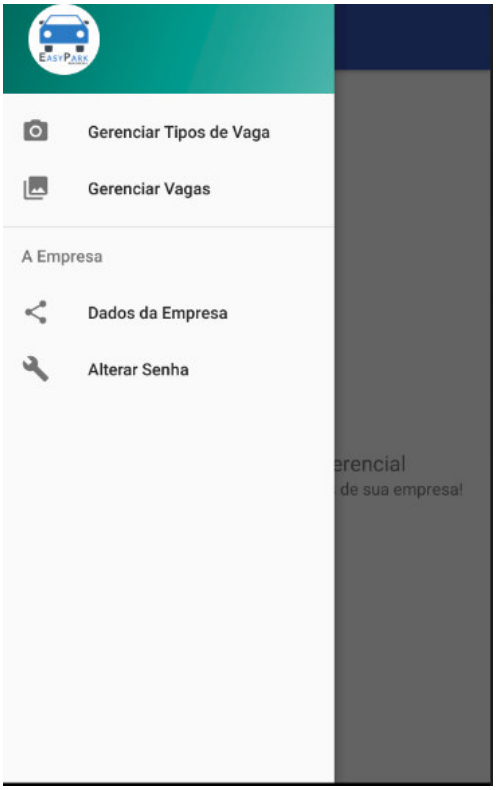

Figura 9: Menu EasyPark Gerencial

A Figura 10 é a representação da tela de gerenciamento de tipos de vaga. Essa tela poderá ser acessada clicando no menu Gerenciar Tipos de Vaga, nela estão listados todos os tipos de vagas já cadastrados no sistema. Clicando sobre uma das opções da lista o usuário poderá fazer a alteração dos dados da mesma, também há opção de fazer o cadastro de um novo tipo clicando no botão Adicionar.

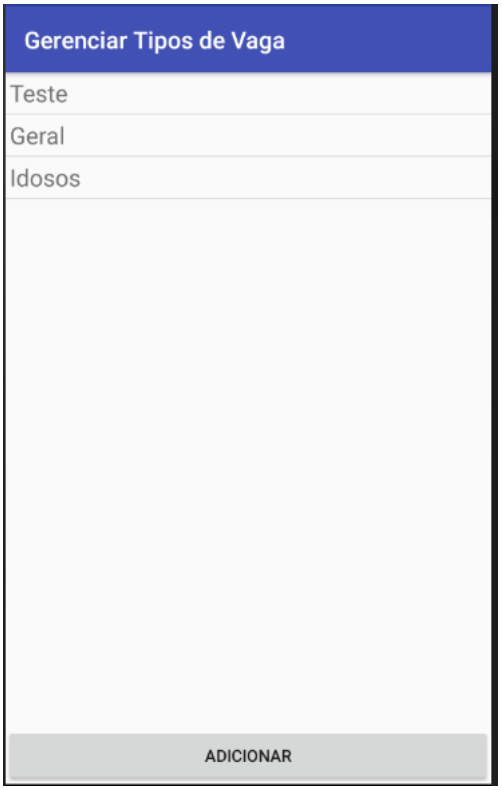

Figura 10: Tela de gerenciamento de tipos de vaga EasyPark Gerencial

A Figura 11 demonstra a tela de cadastro de tipos de vaga. Essa tela poderá ser acessada clicando no botão Adicionar na tela de gerenciamento de tipos de vaga. Nessa tela o usuário preencherá o campo nome e efetuará o cadastro.

Figura 11: Tela de cadastro de tipos de vaga EasyPark Gerencial

A Figura 12 demonstra a tela de gerenciamento de vagas. Essa tela poderá ser acessada clicando no menu Gerenciar vagas, nela estão listadas todas as vagas já cadastradas no sistema. Clicando sobre uma das opções da lista o usuário poderá fazer a alteração dos dados da mesma, também a há opção de fazer o cadastro de uma nova vaga clicando no botão Adicionar.

Figura 12: Tela de gerenciamento de vagas EasyPark Gerencial

A Figura 13 demonstra a tela de cadastro de vagas, ela poderá ser acessada clicando no botão Adicionar na tela de gerenciamento de vagas. Nessa tela o usuário preencherá os campos informados na tela e efetuará o cadastro.

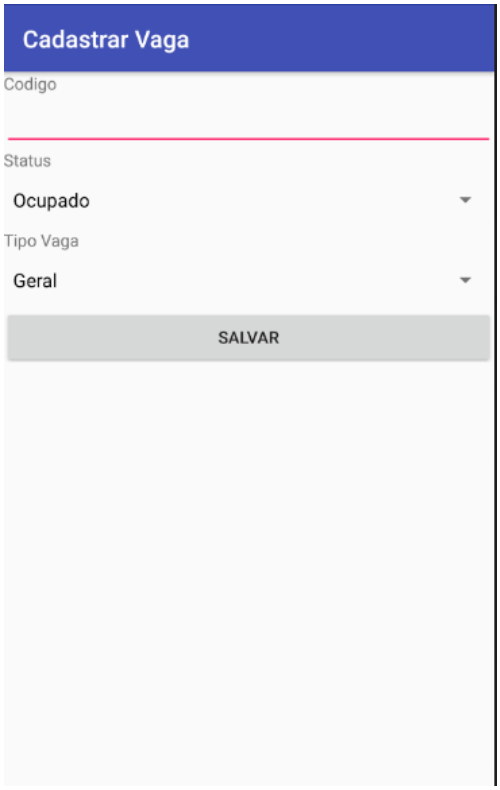

Figura 13: Tela de cadastro de vagas EasyPark Gerencial

A Figura 14 demonstra a tela dos dados do estabelecimento. Essa tela poderá ser acessada clicando no menu Dados da Empresa. Nela o usuário poderá verificar todos os dados do estabelecimento, que já vem previamente preenchidos conforme consta no bando de dados, e fazer alteração conforme o necessário.

Figura 14: Tela de dados da empresa EasyPark Gerencial

A Figura 15 demonstra a tela e alteração de senha, que poderá ser acessada clicando no menu Alterar Senha. Nela o usuário poderá fazer o cadastro de uma nova senha para acesso ao sistema, para tal deve-se informar a senha atual e preencher a nova senha.

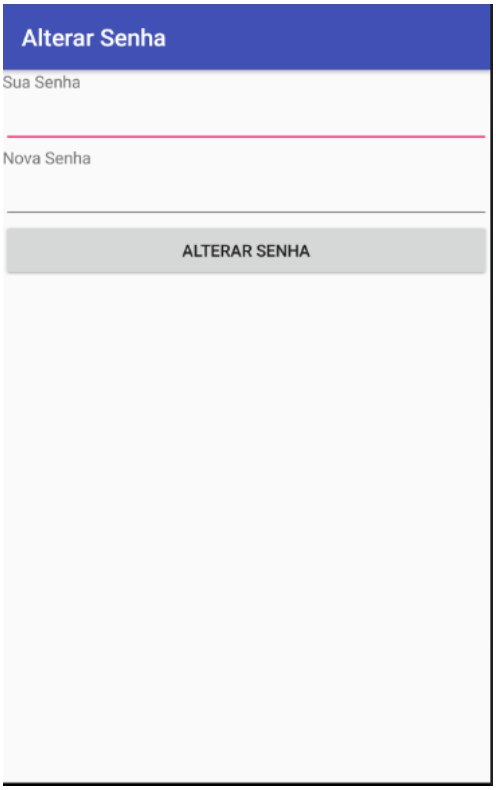

Figura 15: Tela de alteração de senha EasyPark Gerencial

4.3.2 EasyPark

A Figura 16 é a representação da tela inicial do aplicativo. A tela inicial apresenta uma listagem dos estabelecimentos cadastrados no sistema. Clicando sobre um dos estabelecimentos abrirá a tela de visualização de vagas.

Figura 16: Tela inicial EasyPark

A Figura 17 é a representação da de visualização de vagas do aplicativo, que poderá ser acessada clicando sobre um dos estabelecimentos listados na tela inicial. A tela apresenta uma listagem de todas as vagas do estabelecimento selecionado, bem como seus respectivos códigos, tipos e *status*.

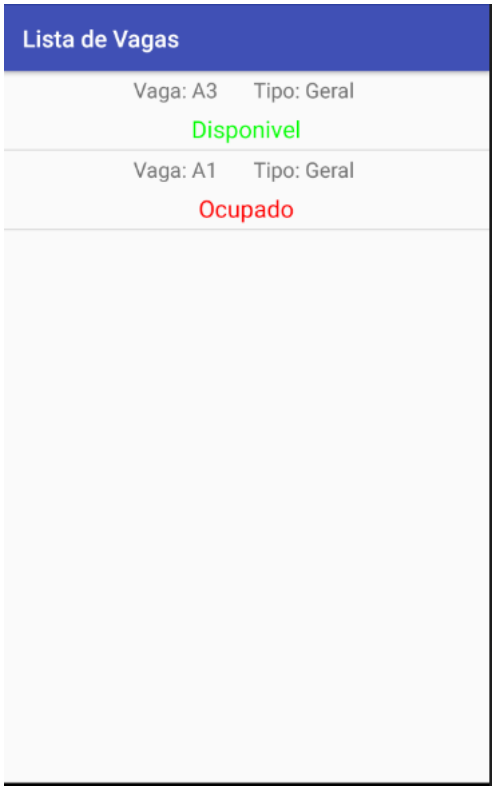

Figura 17: Tela de visualização de vagas EasyPark

4.4 IMPLEMENTAÇÃO DO SISTEMA

A seguir é feita a demonstração do modo de montagem e desenvolvimento da parte hardware do sistema, envolvendo a plataforma Raspberry Pi e o sensor de proximidade.

A Figura 18 a seguir demonstra o esquema de ligação do sensor ultrassônico hc-sr04 ao Raspberry Pi. Para o desenvolvimento do projeto foram utilizados um Raspberry Pi model B, um sensor ultrassónico hc-sr04, uma *protoboard*, dois resistores 1k, quatro *jumpers* macho-fêmea e dois *jumpers* macho-macho.

O *jumper* vermelho representado na Figura 18 é conectado ao pino 5V do Raspberry Pi, o mesmo é distribuído até a *protoboard*, em que é realizada a conexão até o pino VCC do sensor. Da mesma maneira o *jumper* preto é conectado ao GND do Raspberry Pi e também distribuído a *protoboard*, então, é feita a conexão até o pino GND do sensor.

O *jumper* amarelo é conectado do pino 12 do Raspberry Pi diretamente ao pino TRIG do sensor. Por fim o *jumper* verde é conectado do pino 18 do Raspberry Pi à *protoboard*, ele então recebe a conexão de um resistor vindo da coluna de pinos GND da *protoboard*, e de outro resistor vindo do pino ECHO do sensor. A ligação do pino ECHO do sensor é feita por

meio de resistores devido ao fato de que esse pino trabalha em uma voltagem menor ao restante do sensor.

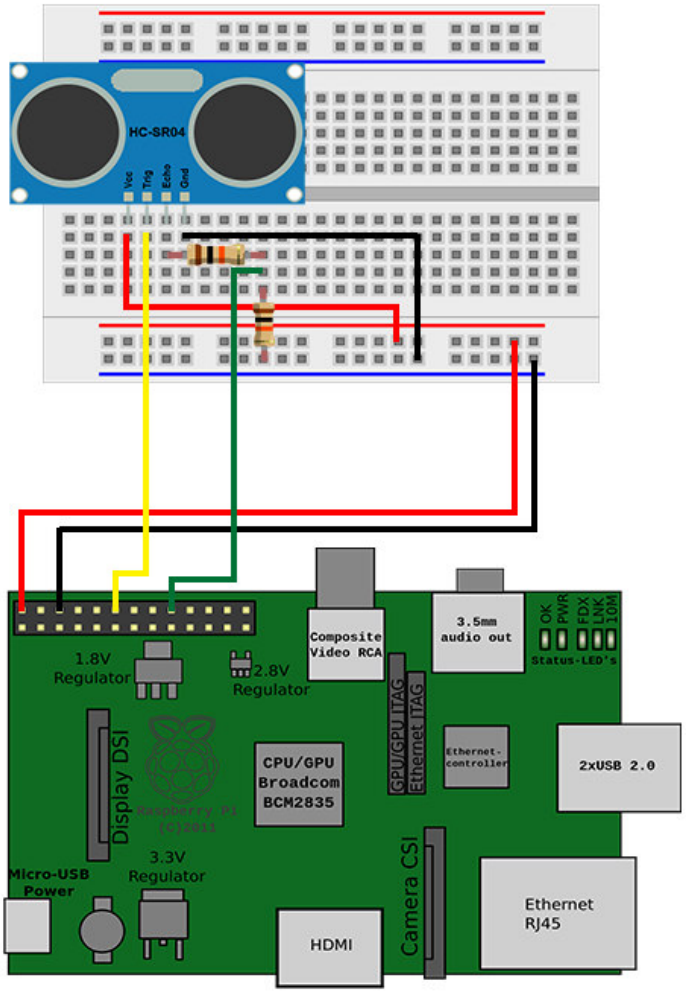

Figura 18: Esquema de montagem sensor ultrassônico hc-sr04 e Raspberry Pi

Durante a implementação do projeto foi perceptível que há uma margem de erro na obtenção das distâncias, devido a fatores externos que influenciam no envio dos sinais ultrassônicos pelo sensor, e pelo fato do sensor ser utilizado apenas para prototipagem e consequentemente não conseguir medidas totalmente precisas.

A seguir são apresentadas partes da codificação do sistema. Essas listagens de código são apresentadas com o objetivo de mostrar como a implementação do sistema foi organizada.

A Listagem 1 representa a codificação desenvolvida em Python na plataforma Raspberry Pi para controle do sensor de proximidade, fazer a conexão com o Firebase e alterar o *status* da vaga no banco de dados. Nessa listagem é feita a conexão junto ao Firebase e obtida como referência uma vaga do banco de dados. Após obter a referência do banco de

dados é realizada a obtenção da distância do objeto em relação ao sensor, a distância é verificada afim de atualizar o status da vaga, se menor que cinquenta centímetros o novo status será Ocupado, caso contrário Disponível. Essa medida de cinquenta centímetros foi utilizada durante o desenvolvimento do projeto, levando em consideração a margem de erro existente na obtenção das distancias pelo sensor.

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
import RPi.GPIO as GPIO
import time

def init_sensor(ref):
    GPIO.setmode(GPIO.BCM)
    GPIO_TRIGGER = 12
    GPIO_ECHO    = 18
    GPIO.setwarnings(False)
    GPIO.setup(GPIO_TRIGGER,GPIO.OUT)
    GPIO.setup(GPIO_ECHO,GPIO.IN)
    GPIO.output(GPIO_TRIGGER,False)

    try:
        while True:
            GPIO.output(GPIO_TRIGGER,True)
            time.sleep(0.01)
            GPIO.output(GPIO_TRIGGER,False)
            start = time.time()
            while GPIO.input(GPIO_ECHO)==0:
                start = time.time()
            while GPIO.input(GPIO_ECHO)==1:
                stop = time.time()
            elapsed = stop-start
            distance = (elapsed * 34300)/2
            print(distance)
            if (distance > 50):
                ref.update({'status': 'Disponivel'})
            else:
                ref.update({'status': 'Ocupado'})
            time.sleep(1)
    except KeyboardInterrupt:
        print("sair")
        GPIO.cleanup()

def main():
    cred = credentials.Certificate('/home/pi/Desktop/projeto/projetotcc-
1ba15-firebase-adminsdk-879bk-de7b1e21ea.json')
```

```
    firebase_admin.initialize_app(cred, {
        'databaseURL': 'https://projetotcc-1ba15.firebaseio.com/'
    })

    ref = db.reference('Vagas/vaga_teste')
    print(ref.get())
    print ('Ok !')
    init_sensor(ref)

if __name__ == '__main__':
    main()
```

Listagem 1: Código Python controle sensor e conexão Firebase

A Listagem 2 representa a codificação desenvolvida em Java para o aplicativo EasyPark, que é responsável pela consulta dos estabelecimentos cadastrados no banco de dados do Firebase, e a apresentação dos mesmos em um ListView. O evento onDataChange() é disparado na primeira conexão com o banco, e toda vez que há uma alteração na instância empresas no banco de dados do Firebase.

```
lvEmpresas = findViewById(R.id.lvEmpresas);

mDatabase = FirebaseDatabase.getInstance().getReference();
mDatabase.child("empresas").addValueEventListener(new ValueEventListener()
{
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        empresas = new ArrayList<>();
        for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {
            if (postSnapshot != null && postSnapshot.exists()) {
                Empresa empresa = postSnapshot.getValue(Empresa.class);
                empresa.setKey(postSnapshot.getKey());
                empresas.add(empresa);
            }
        }

        if (empresas != null) {
            AdapterEmpresa adapter = new
AdapterEmpresa(PrincipalActivity.this, empresas);
            lvEmpresas.setAdapter(adapter);
        }
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.d("PrincipalActivity", databaseError.toString());
    }
});
```

Listagem 2: Consulta de estabelecimentos e apresentação em ListView

A Listagem 3 representa o código desenvolvido no aplicativo EasyPark para realizar a consulta das vagas relativas a empresa selecionada no banco de dados do Firebase, e fazer a apresentação dos dados obtidos em um ListView. Na abertura da tela uma referência contendo o *id* do estabelecimento é recebida. Utilizando essa referência é feita a consulta no banco de dados do Firebase para obtenção de todas as vagas relativas ao estabelecimento em questão.

```
lvPrincipal = findViewById(R.id.lvPrincipal);

mDatabase = FirebaseDatabase.getInstance().getReference();

Bundle extras = getIntent().getExtras();
if (extras != null) {
    keyEmpresa = extras.getString("KEY"); //id do estabelecimento para
                                          // filtragem das vagas

}

mDatabase.child("Vagas").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        vagas = new ArrayList<>();
        for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {
            if (postSnapshot != null && postSnapshot.exists()) {
                Vaga vaga = postSnapshot.getValue(Vaga.class);
                if (vaga.getEmpresa().equals(keyEmpresa)) {
                    vaga.setKey(postSnapshot.getKey());
                    vagas.add(vaga);
                }
            }
        }

        if (vagas != null) {
            AdapterVaga adapter = new AdapterVaga(VagasActivity.this,
vagas);
            lvPrincipal.setAdapter(adapter);
        }
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.d("VagasActivity", databaseError.toString());
    }
});
```

Listagem 3: Consulta de vagas e apresentação em ListView

A Listagem 4 representa o método btSalvarVagaOnClick desenvolvido para o aplicativo EasyPark Gerencial, que tem como finalidade realizar o cadastro de uma nova vaga no banco de dados do Firebase.

```
public void btSalvarVagaOnClick(View view) {
    if (etCodigo.getText().toString().isEmpty()) {
```

```
        Toast.makeText(FormVagaActivity.this, "Preencha o campo Código!",
Toast.LENGTH_SHORT).show();
        etCodigo.requestFocus();
    } else {
        vaga.setCodigo(etCodigo.getText().toString());
        vaga.setStatus(String.valueOf(spStatus.getSelectedItem()));
        vaga.setTipoVaga(String.valueOf(spTipoVaga.getSelectedItem()));
        vaga.setEmpresa(empresaKey);

        Map<String, Object> atualizacoes = new HashMap<>();
        atualizacoes.put("/Vagas/" + key, vaga);
        mDatabase.updateChildren(atualizacoes).addOnSuccessListener(new
OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                finish();
                Toast.makeText(FormVagaActivity.this, "Salvo com
Sucesso!", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Listagem 4: Método btSalvarVagaOnclick

A Listagem 5 representa o método remover desenvolvido para o aplicativo EasyPark Gerencial, que tem como finalidade realizar a remoção de uma vaga no banco de dados do Firebase.

```
public void remover() {
    AlertDialog alertDialog = new
AlertDialog.Builder(FormVagaActivity.this).create();
    alertDialog.setTitle("Atenção");
    alertDialog.setMessage("Deseja remover o registro?");
    alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "Não",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            });
    alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "Sim",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    Map<String, Object> atualizacoes = new HashMap<>();
                    atualizacoes.put("/Vagas/" + key, null);

mDatabase.updateChildren(atualizacoes).addOnSuccessListener(new
OnSuccessListener<Void>() {
                        @Override
                        public void onSuccess(Void aVoid) {
                            finish();
                            Toast.makeText(FormVagaActivity.this,
"Registro Removido!", Toast.LENGTH_LONG).show();
                        }
                    });
                }
            });
    alertDialog.show();
}
```

Listagem 5: Método remover Vaga

A Listagem 6 representa o método btSalvarEmpresaOnClick desenvolvido para o aplicativo EasyPark Gerencial, que tem como finalidade salvar as alterações feitas nos dados do estabelecimento no banco de dados do Firebase.

```
public void btSalvarEmpresaOnClick(View view) {
    Empresa empresa = new Empresa();
    boolean nomeOk = true;
    boolean cnpjOk = true;
    boolean emailOk = true;
    CnpjValido cnpjValido = new CnpjValido();

    if (etNome.getText().toString().isEmpty()) {
        Toast.makeText(EmpresaActivity.this, "Preencha o campo Nome!",
Toast.LENGTH_SHORT).show();
        etNome.requestFocus();
        nomeOk = false;
    }

    if (etCnpj.getText().toString().isEmpty()) {
        Toast.makeText(EmpresaActivity.this, "Preencha o campo CNPJ!",
Toast.LENGTH_SHORT).show();
        etNome.requestFocus();
        cnpjOk = false;
    } else if (!cnpjValido.isCnpjValido(etCnpj.getText().toString())) {
```

```
        Toast.makeText(EmpresaActivity.this, "CNPJ inválido!",
Toast.LENGTH_SHORT).show();
        etNome.requestFocus();
        cnpjOk = false;
    }

    if (etEmail.getText().toString().isEmpty()) {
        Toast.makeText(EmpresaActivity.this, "Preencha o campo E-mail!",
Toast.LENGTH_SHORT).show();
        etNome.requestFocus();
        emailOk = false;
    } else if (!etEmail.getText().toString().contains("@")) {
        Toast.makeText(EmpresaActivity.this, "E-mail inválido!",
Toast.LENGTH_SHORT).show();
        etNome.requestFocus();
        emailOk = false;
    }

    if (nomeOk && cnpjOk && emailOk) {
        empresa.setNome(etNome.getText().toString());
        empresa.setCnpj(etCnpj.getText().toString());
        empresa.setEmail(etEmail.getText().toString());
        empresa.setEndereco(etEndereco.getText().toString());
        empresa.setBairro(etBairro.getText().toString());
        empresa.setCidade(etCidade.getText().toString());
        empresa.setEstado(etEstado.getText().toString());
        empresa.setSenha(senha);

        Map<String, Object> atualizacoes = new HashMap<>();
        atualizacoes.put("/empresas/" + key, empresa);
        mDatabase.updateChildren(atualizacoes).addOnSuccessListener(new
OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                finish();
                Toast.makeText(EmpresaActivity.this, "Salvo com Sucesso!",
Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Listagem 6: Método btSalvarEmpresaOnClick

5 CONCLUSÃO

O objetivo deste trabalho foi efetuar o desenvolvimento inicial de um sistema para ser executado em dispositivos móveis que possibilite a automação das vagas de estacionamento de comércios em geral. O aplicativo tem o objetivo de facilitar aos clientes a forma de verificar locais com vagas disponíveis para estacionar seus automóveis, evitando que o gasto de um tempo considerável até encontrar uma vaga disponível.

As tecnologias utilizadas são as plataformas Raspberry Pi e Android. O sistema foi dividido em dois aplicativos, um visando o controle dos cadastros dos dados, e outro dedicado ao usuário. Os aplicativos são utilizados pelo funcionário do estabelecimento e pelo cliente.

Após a realização da definição dos casos de usos e do diagrama de classes foi desenvolvido o aplicativo EasyPark Gerencial, neste aplicativo é possível fazer o cadastro dos dados referentes aos tipos de vaga, as vagas e os dados do estabelecimento. Após foi realizado o desenvolvimento do aplicativo EasyPark que dá a visão dos estabelecimentos e suas vagas ao cliente, e feito o desenvolvimento da codificação no Raspberry Pi para controle dos sensores e conexão como Firebase.

A realização deste trabalho possibilitou um grande aprendizado, tanto na utilização da plataforma Raspberry Pi quanto no desenvolvimento do aplicativo. Como maiores dificuldades encontradas no desenvolvimento do projeto, estão a tentativa frustrada de utilização da plataforma Arduino para conexão com o banco de dados em nuvem, mais precisamente a utilização do módulo Wi-Fi ESP8266 ligado ao Arduino, e a utilização do micro controlador ESP8266 NodeMCU para a determinada função.

Como trabalhos futuros estão previstos a utilização da classe *canvas* do Android para a criação de um mapa do estacionamento do estabelecimento, contendo todas as vagas do mesmo. Também está previsto a utilização da localização do cliente para a filtragem dos estabelecimentos mais próximos ao local em que o mesmo se encontra.

REFERÊNCIAS

ANDROID DEVELOPERS. Disponível em: <https://developer.android.com/about/dashboards/index.html> Acesso em: 10 nov. 2017.

DA SILVA, Luciano Alves. **A Apostila de Android – Programando Passo a Passo.** Disponível em: <http://othonbatista.com/arquivos/android/apostila-android.pdf>. Acesso em: 10 nov. 2017.

FILIPEFLOP. **Como Conectar O Sensor Ultrassônico Hc-Sr04 Ao Arduino**. Disponível em: <https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/>. Acesso em: 26 nov. 2018.

FIREBASE. Disponível em: <https://firebase.google.com/support/faq/?hl=pt-br>. Acesso em: 05 nov. 2017.

MONK, Simon. **Programando o Raspberry Pi**. São Paulo: Novatec, 2013.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software-8ª Edição.** McGraw Hill Brasil, 2016.

PYTHON. Disponível em: <https://www.python.org/> Acesso em: 10 nov. 2018.

RASPBERRY PI. Disponível em: < https://www.raspberrypi.org/downloads/>. Acesso em: 05 nov. 2018.

RELATÓRIO DA FROTA CIRCULANTE 2017. Disponível em: < http://www.sindipecas.org.br/sindinews/Economia/2017/R_Frota_Circulante_2017.pdf>. Acesso em: 26 ago. 2017.

VISUAL PARADIGM. Disponível em: <https://www.visual-paradigm.com/support/faq.jsp>. Acesso em: 05 nov. 2017.

ZURIARRAIN, José Mendiola. **Android já é o sistema operacional mais usado do mundo.** Disponível em: <https://brasil.elpais.com/brasil/2017/04/04/tecnologia/1491296467_396232.html> Acesso em: 27 ago. 2017.