GUILHERME JOB ROCHA SILVEIRA

SISTEMA DE GERENCIAMENTO E AUTOMAÇÃO DE ESTACIONAMENTOS DE SHOPPING CENTERS ORIENTADO A MICROSSERVIÇOS E *CLOUD COMPUTING*

SÃO PAULO

2016

GUILHERME JOB ROCHA SILVEIRA

SISTEMA DE GERENCIAMENTO E AUTOMAÇÃO DE ESTACIONAMENTOS DE SHOPPING CENTERS ORIENTADO A MICROSSERVIÇOS E *CLOUD COMPUTING*

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de graduado em Engenharia de Computação

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. Jorge Luis Risco Becerra

SÃO PAULO

2016

Dedico este trabalho aos meu pais, Eduardo e Regina, pelo apoio, confiança e compreensão nas escolhas que fiz que me fizeram tornar quem sou.

Ao meu padrasto, Duarte, pela orientação e planejamento para me propiciar o ambiente onde consegui minhas maiores conquistas.

Aos amigos e colegas que me ajudaram e acompanharam durante toda a minha trajetória na graduação.

AGRADECIMENTOS

Ao professor Jorge Luis Risco Becerra, pela paciência, confiança e orientação em todos os momentos durante o ano.

Ao Leonardo Ogura Martins, aluno do curso de Engenharia Mecatrônica, por usar o seu trabalho de conclusão de curso o qual, juntamente com este, torna-se um projeto maior – a Parkaware.

A Escola Politécnica da Universidade de São Paulo, que nos forneceu a infraestrutura e condições para a realizações deste trabalho.

Ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS), por oferecer um curso de excelência que foi base para a criação deste trabalho.

"A tecnologia torna as grandes populações possíveis;
grandes populações tornam a tecnologia indispensável. "

(Joseph Krutch)

RESUMO

Um problema enfrentado em shopping centers é o desconforto que o estacionamento gera para seus usuários, desde a entrada retirando o tíquete da cancela até a hora do pagamento nos balcões reservados. O fluxo do motorista ter que pegar o tíquete da cancela, o guardar, depois pegar fila para realizar o pagamento e, finalmente, o inserir na cancela de saída pode propiciar estresse, gerado por acasos como esquecer onde pôs o tíquete, pegar fila para realizar o pagamento, o cartão não ter limite disponível ou não aceitar sua bandeira, o caixa não ter troco. Além disso, o estacionamento pode estar tão cheio que não dê para o motorista encontrar um lugar para estacionar o carro. Para solucionar tais desconfortos, este trabalho propõe a utilização de um aplicativo móvel para efetuar consulta de disponibilidades do estacionamento, além de automatizar o processo do pagamento das estadias. Para isso, este trabalho foi desenvolvido a partir de metodologias empregadas no curso de Engenharia de Computação da Escola Politécnica da Universidade de São Paulo, somado ao uso de tecnologias emergentes nos mercado e pesquisa. Este documento apresenta sugestões sobre como esse projeto pode ser aprimorado no futuro.

Palavras-chave: Estacionamentos. Microsserviços. *Cloud Computing*. Shopping Center. Programação Funcional

ABSTRACT

A recurring problem faced on mall is users discomfort propitiated by parking lot, since getting ticket at entrance until paying it at payment balcony. To solve such discomfort this work proposes the use of mobile application to make mall available searches, besides of make staying payments automatic. In order to make it, this work was made from Computer Engineering classes and growing technologies in market and research. This work gives suggestions about how this can be upgraded in near future.

Key-words: Parking Lot. Microservices. Cloud Computing. Shopping Center. Functional Programming

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AWC	Academic Working Capital
AWS	Amazon Web Services
CRUD	Create, Read, Update and Delete
DevOps	Development and Operations
DNS	Dynamic Domain Name System
HTTP	Hyper-Text Transfer Protocol
HTTPS	Hyper-Text Transfer Protocol Secure
IAM	Identity and Access Management
ICR	Intelligent Character Recognition
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model-View-Controller
NoSQL	No Structured Query Language
OCR	Optical Character Recognition
PCS	Departamento de Engenharia de Computação e Sistemas Digitais
PMR	Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos
RENAVAM	Registro Nacional de Veículos Automotores
REST	Representational State Transfer
RESTful	Representational State Transfer
RFID	Radio-Frequency Identification

SOA	Service-Oriented Architecture
SQL	Structured Query Language
SRPLV	Sistema de Reconhecimento da Placa de Licenciamento Veicular
URL	Uniform Resource Locator
USP	Universidade de São Paulo
VM	Virtual Machine
WLAN	Wireless Local Area Network
WPS	Wi-Fi Protected Setup
WS	Windows Service

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

1.1. DEFINIÇÃO DO PROBLEMA

Um problema recorrente ao sair de casa de automóvel é o estacionamento. O estresse e desconforto de ter que dispor de muito tempo para achar uma vaga para estacionar seu veículo e o fluxo do tíquete imposto pelo estacionamento podem gerar muitas reclamações e insatisfação das pessoas.

Com isso, a demanda por opções de estacionamentos capazes de resolver este problema de uma forma hábil, segura e correta tem crescido ao longo dos anos. Adicionando os fatos que o número de carros vem crescendo ao longo dos dias, (RODRIGUES, 2013), presume-se que problemas de estacionamentos estarão por aumentar.

Logo, o desenvolvimento de alternativas inteligentes, não custosas e eficientes são fatores diferenciais para a solução do grande problema do tempo gasto do cliente em achar uma vaga disponível e assim estacionar seu veículo.

Estacionamentos inteligentes são soluções tecnológicas para o problema da organização em geral de estacionamentos. Já existem estacionamentos que indicam o número de vagas por andar, e utilizam sinais luminosos para indicar quais vagas estão disponíveis. Também existem estacionamentos totalmente autônomos em que o usuário apenas deixa seu carro em uma plataforma e assim um sistema automático estaciona o veículo no interior de um edifício, bem como trazendo-o de volta ao usuário. (ALHAK, S.H.A, 2011).

Resumindo, o problema claro em estacionamentos de locais com uma grande quantidade de automóveis é a dificuldade de identificar uma vaga livre como até mesmo saber se o estacionamento está superlotado antes de entrar neste, além da manipulação do tíquete não atender as exigências atuais de comodidade dos motoristas.

1.2. OBJETIVO

O objetivo deste trabalho é desenvolver um sistema de gerência e automação para estacionamentos de shopping centers que consiste em um aplicativo móvel para iPhone e uma

plataforma web, utilizando conceitos da arquitetura de microsserviços para a arquitetura do sistema e hospedando todos os servidores e a plataforma web na *cloud*.

Resumidamente podemos citar as cinco funções principais desse sistema:

1. Entrada e saída automática no estacionamento, ou seja, a cancela abre automaticamente para veículos cadastrados no sistema. Assim, o sistema é capaz de realizar o pagamento automaticamente via aplicativo celular.

2. Descobrir o número de vagas disponíveis em um estacionamento. Assim o usuário poderá ter acesso ao estado em que o estacionamento se encontrar, podendo inferir o quão lotado o local se encontra.

3. Localizar a vaga onde o usuário estacionou seu carro. Após terminar suas compras o usuário poderá utilizar o sistema para descobrir aonde seu carro foi estacionado, função bastante útil para estacionamentos que comportam um grande volume de veículos.

4. Reservar previamente vagas no estacionamento. O usuário poderá garantir uma vaga previamente utilizando o sistema, função bastante requerida no caso de datas cuja as quais as vagas dos estacionamentos dos estabelecimentos são bastantes requisitadas.

5. Receber descontos e promoções especiais de lojistas cujo o qual o estacionamento faz parte de suas lojas. Uma forma de conciliar a ânsia dos lojistas de compartilhar suas promoções com a vontade dos usuários de encontrarem os melhores produtos nos melhores preços e ainda receberem desconto na fatura do estacionamento.

O sistema possui funcionalidades de gerência para o responsável pelo estacionamento e informações de informação e automação para os motoristas.

1.3. MOTIVAÇÃO ACADÊMICA

Por motivação acadêmica, tem-se o objetivo de aprender tecnologias modernas que começaram a ser empregadas no mercado e na pesquisa, tanto voltado para a arquitetura quanto para soluções de software, desde a escolha da linguagem até a estrutura do código.

Esse trabalho tem, em essência, viés arquitetural, onde o principal foco é a arquitetura em microsserviços. A arquitetura de microsserviços (FOWLER, M. 2014) é, além de uma arquitetura de sistema, uma arquitetura empresarial com estudo desde a criação de equipes até a arquitetura de software, por isso, seu estudo é amplo e será mostrado nesse projeto.

Concomitante à arquitetura, o projeto tem o conceito de *cloud computing* integrado. O *cloud computing*, que é uma tecnologia muito importante atualmente pela redução de custos e a facilidade de escalar infraestrutura de projetos, será detalhado ao decorrer desta monografia.

O uso de linguagens novas como Clojure, Swift e o *framework* Angular também foram empregados no projeto. A linguagem de programação Clojure é muito importante atualmente por ser uma linguagem funcional. Estas estão voltando a ganhar espaço no mercado pela necessidade de manter os dados imutáveis, na seção 5.3.2 abordamos a necessidade de se empregar linguagens funcionais no atual cenário desenvolvimento de softwares orientados a multiprocessadores.

1.4. MOTIVAÇÃO PROFISSIONAL

Esse trabalho é justificado pela finalidade de se tornar um produto empresarial. Ao término, o produto resultante do estudo e desenvolvimento será usado em prol da empresa **Parkaware Tecnologia Ltda**.

Este trabalho é uma parte de um projeto maior com uma perspectiva de se tornar um produto no futuro. O mesmo está sendo coordenado e acompanhado pelo AWC (*Academic Working Capital*) do instituto TIM. Este programa, com um viés filantrópico, que auxilia estudantes universitários interessados em conceber seu negócio de base tecnológica nas áreas de Engenharias e Ciências Exatas. O programa auxilia principalmente no ponto de vista empreendedor do projeto.

A Parkaware Tecnologia Ltda é uma empresa criada pelo autor, um colega de turma da graduação e mais um integrante do último ano do curso de engenharia de mecatrônica, com o intuito de proporcionar soluções para estacionamentos. Atualmente, a empresa participa do Academic Working Capital – AWC do **Instituto TIM**, que arca com os custos operacionais e auxiliam na visão estratégica da empresa. Este programa, com um viés filantrópico, auxilia estudantes universitários interessados em conceber seu negócio de base tecnológica nas áreas de Engenharias e Ciências Exatas. O programa auxilia principalmente no ponto de vista empreendedor do projeto. Além disso, a empresa se encontra incubada no Centro de Inovação, Empreendedorismo e Tecnologia – o **CIETEC**, o qual fornece um espaço físico colaborativo, auxiliam na visão estratégica da empresa e mercado e facilita o *networking*.

Na visão do autor da dissertação, criar uma solução com tecnologia moderna é essencial para que as empresas cresçam com menos atritos propiciados por limites de

escalabilidade e equipes de desenvolvimento, onde, ao se adotar as novas tecnologias, a empresa favorece em reter funcionários sedentos por conhecimento e dispostos e aprender como a tecnologia pode se reciclar dentro da empresa, contribuindo para um ciclo virtuoso.

1.5. DIVISÃO INTERDEPARTAMENTAL

O **Sistema de Gerenciamento e Automação de Estacionamentos Orientado à Microsserviços e *Cloud Computing*** é parte de um projeto chamado de **Parkaware**. A **Parkaware**, uma fusão de "estacionar" com "estar ciente", em inglês, é um sistema de gerenciamento de estacionamentos via reconhecimento de imagens, onde através deste sistema o usuário consegue saber em qual vaga está estacionado o seu carro, e consultar previamente a lotação das vagas de um estabelecimento. Também é possível que o cliente consiga reservar uma vaga e empresas próximas anunciem mercadorias pelo aplicativo. Ademais, com o **Parkaware**, é possível gerenciar o sistema de controle do tempo de permanência assim como o pagamento do estacionamento automaticamente.

Do ponto de vista acadêmico, a construção deste projeto foi divido em dois trabalhos de conclusão de curso. Um para o Departamento de Engenharia de Computação e Sistemas Digitais (PCS) e o outro para o Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR), ambos da Escola Politécnica da Universidade de São Paulo.

O integrante do departamento de engenharia da computação ficará responsável pelo desenvolvimento do aplicativo piloto para smartphones que irá interagir com o usuário, além de plataforma web. No aplicativo o usuário poderá ter acesso à quantidade de vagas disponíveis do estacionamento, e realizar o pagamento automático para o estacionamento. Na aplicação web o gestor do estacionamento poderá ter acesso a funcionalidades de geração de relatórios e emissão de dados estatísticos que auxiliam na decisão estratégica do estacionamento como previsão de orçamento e gráficos diversos.

Adicionalmente, o projeto de engenharia de computação tem caráter arquitetural, focando em arquitetura de microsserviços (MARTIN FOWLER, 2013). Com isso, apresentamos as principais funcionalidades abordadas pela equipe de computação, as quais serão desenvolvidas nesta monografia. Para adquirir dados, o sistema conta com a integração com a aplicação local, que ficará anexada aos estacionamentos, responsável pelos integrantes de engenharia de mecatrônica e explanado a seguir.

Os integrantes do departamento de engenharia mecatrônica ficarão responsáveis pelo mapeamento das vagas de estacionamentos utilizando câmeras. As mesmas funcionarão continuamente à procura de vagas livres no estacionamento, para assim atualizar o estado do mesmo. Um algoritmo de identificação de placas irá constantemente procurar uma placa na região das vagas estudadas, no caso de sucesso adicionamos ao nosso banco de dados a placa identificada ligada à vaga em que a mesma se encontra e atualizamos o estado da vaga para indisponível, no caso de não encontrarmos, atualizaremos o estado da vaga para disponível. Tais informações serão úteis para o aplicativo para smartphones que será explicada mais à frente.

Além disso, a equipe de mecatrônica irá realizar o reconhecimento do veículo logo na entrada do estacionamento e na saída do mesmo, sendo possível assim realizar o sistema de gerenciamento do pagamento do tempo de permanência do veículo no estacionamento, não necessitando de tickets ou outros dispositivos instalados no carro do usuário.

A estrutura geral resumida do projeto **Parkaware** é ilustrado na Figura 1, mostrando como as partes do projeto serão integradas, e a divisão de cada projeto de conclusão de curso, designando a parte para cada departamento nas Figuras 2 e 3.

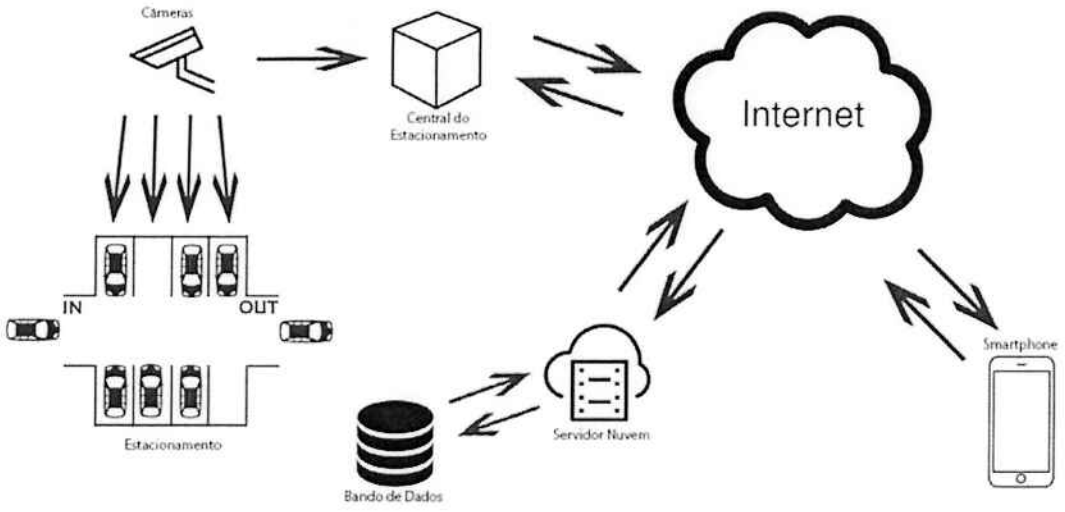

Figura 1 - Estrutura geral do sistema Parkaware

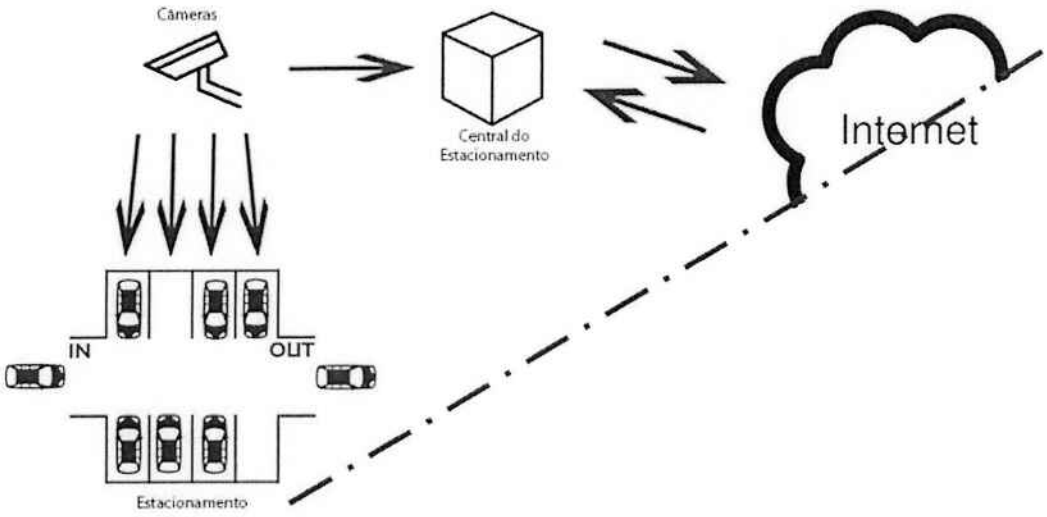

Figura 2 - Parte designada à equipe de Engenharia de Mecatrônica

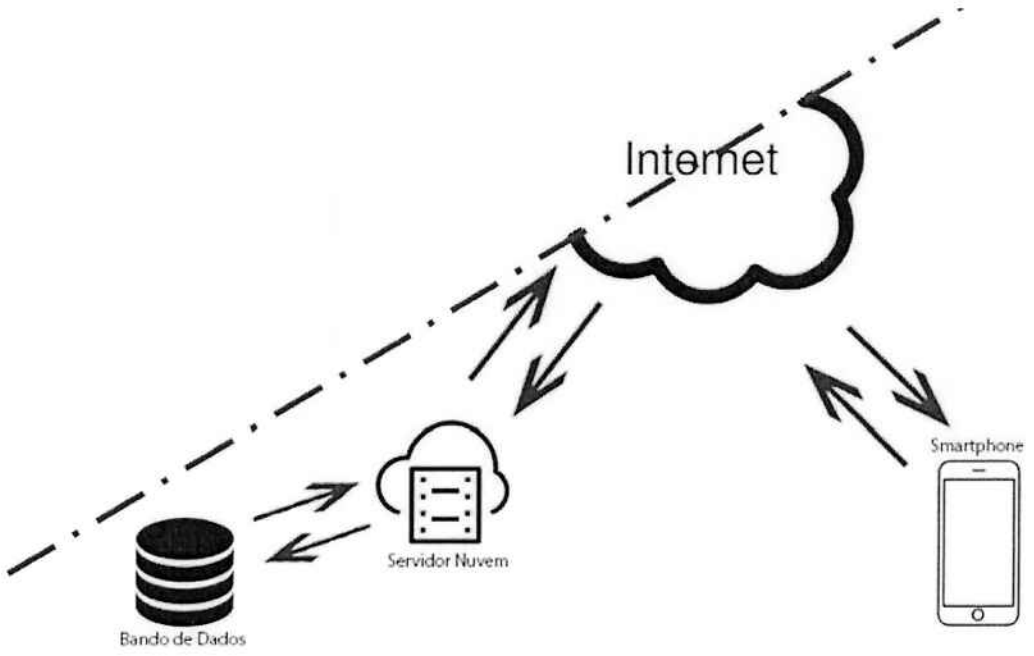

Figura 3 - Parte designada à equipe de Engenharia de Computação

1.6. JUSTIFICATIVA

Primeiramente, para um projeto de microsserviços, é necessário saber o que é um microsserviço. Para isso, usamos abordagens descritas conforme o livro de (NEWMAN S.; 2015).

De acordo com o livro da IBM, de autores (DAYA, S.; DUY, N. D.; EATI, K.; FERREIRA, C. M.; GLOZIC, D.; GUCIR, V.; GUPTA, M.; JOSHI, S.; LAMPKIN, V.; MARTINS M.; NARAIN, S.; VENNAN, R.; 2015) é importante sabermos quando não usar

microsserviços. Daqui, é explicado que devemos tomar cuidado ao elaborar arquitetura de microsserviços:

- Não começar o projeto com microsserviços;
- Não usar microsserviços sem *DevOps*;
- Não gerenciar sua própria infraestrutura;
- Não criar muitos microsserviços;
- Não esquecer de monitorar a latência das requisições;

Para o entendimento e estudo de *DevOps* é usado o livro dos autores (KIM, G.; BEHR, K.; SPAFFORD, G. 2015), que é uma novela explicitando soluções gerais de *DevOps* no meio da tecnologia.

O segundo passo é entender como funciona a linguagem que usamos no desenvolvimento da solução, Clojure. O livro de (HIGGINBOTHAM, D. 2015) é saudoso no sentido de mostrar como a linguagem funcional está inserida no contexto atual da tecnologia.

Por outro lado, para aplicar este conhecimento no cenário de estacionamentos, é importante entender como o uso de câmeras é aplicável no cenário em questão. Na dissertação de (SUSIN, TOZZI e VON ZUBEN, 2005), existe uma distinção entre os diversos métodos de identificação automática de automóveis. Podemos citar quatro maneiras diferentes:

1. Sistemas que utilizam a imagem da placa de licenciamento veicular;
2. Sistemas que utilizam um código via satélite;
3. Sistemas que utilizam um código via radiofrequência;
4. Sistemas que utilizam um código via cartão.

1.7. ORGANIZAÇÃO DO PROJETO

Nesta parte, é explicado, de antemão, como está organizado o projeto nessa monografia.

Na Seção 2 são explanados os principais conceitos e tecnologias envolvidas no desenvolvimento da solução.

Na Seção 3 apresentada e explicada a metodologia que foi utilizada nesse projeto, os principais desafios que compõem a teoria do projeto, e porque essa solução é conveniente e os desafios que impulsionaram o uso de determinados conceitos, como microsserviços e linguagem funcional, por exemplo. E é nesta seção, a mais extensa, que tratamos da solução. Nela, há subseções de especificação de requisitos e arquitetura e detalhes de implementação.

Na Seção 4 é finalizado o trabalho, dando destaque para as conclusões tiradas com ele, as contribuições que esse trabalho trouxe para o integrante e o mercado e os próximos desafios que impulsionarão a continuação do projeto.

2. CONCEITOS E TECNOLOGIAS ENVOLVIDAS

Neste capítulo é apresentado os principais conceitos e tecnologias que estão envolvidos no projeto e são necessárias pleno conhecimento para que o resto do documento seja entendível.

Em 3.1 e 3.2 são apresentados os principais conceitos envolvidos, ao passo que no resto do capítulo são explicitadas as principais tecnologias adotadas na solução.

2.1. CRITÉRIO DE ESCALABILIDADE

A abordagem de microsserviços é injetada no projeto como uma alternativa de prover escalabilidade real do sistema, segundo o cubo de escalabilidade. Essa abordagem se mostra necessária devido ao fato que, atualmente, nos *shopping centers*, uma média de 100 milhões de pessoas vão de carro a este destino. Isso resulta numa taxa de 50 carros entrando no estacionamento por segundo.

Diante disso, o número de requisições ao nosso sistema pode crescer a números estrondosos e, para isso, é importante que o sistema tenha uma maneira simples de aumentar a capacidade de receber requisições. E é nesse contexto que se faz o uso de microsserviços.

2.1.1.1. O Cubo de Escalabilidade

No modelo proposto por (FISHER, M. T.; ABBOTT M. L. 2009) é adotado um modelo de escalabilidade em três dimensões, conforma figura 4.

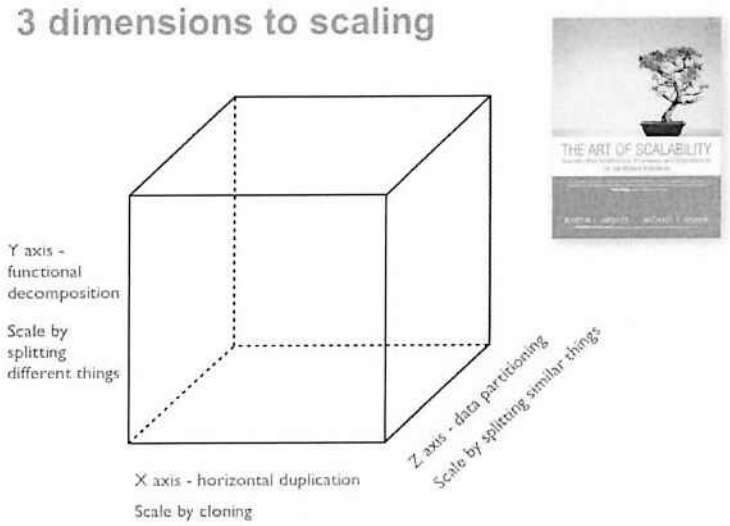

Figura 4 - As três dimensões de escalabilidade

Em cada dimensão, é possível escalar o sistema de uma abordagem diferente:

- No eixo X, que consiste na escalabilidade horizontal, é instanciada múltiplas cópias da mesma aplicação atrás de um distribuidor de carga. Se tiver N cópias de uma aplicação, cada aplicação recebe uma carga 1/N do distribuidor. Esta é uma abordagem simples e muito utilizada atualmente.
- No eixo Y, que consiste na quebra de um projeto grande em serviços menores, cada serviço é responsável por uma ou mais funções corretadas. Existem diversas maneiras de decompor uma aplicação em diferentes serviços. Uma abordagem sugere que usemos um serviço para cada caso de uso. Outra opção é decompor por substantivos, e criar serviços responsáveis por todas as operações, como, por exemplo, "gerenciamento de cliente" e "pagamento". Sendo possível, até, mesclar as abordagens e ter uma decomposição orientada à verbos e substantivos.
- No eixo Z, que é comumente usado para escalar banco de dados, os dados são particionados entre um conjunto de serviços, onde cada serviço manipula uma parte do dado. Uma abordagem que contempla esse tipo de análise é o modelo de escalabilidade *sharding* de banco de dados.

A arquitetura de microsserviços é usada para escalonar a aplicação no sentido Y dessa abordagem. Entretanto, é possível usar os conceitos dos demais eixos com efetividade para que o sistema tenha uma escalabilidade única dentre as arquiteturas vigentes de sistema.

2.2. ARQUITETURA DE MICROSSERVIÇOS

Microsserviços é uma abordagem de arquitetura de software onde o serviço do sistema é dividido vários serviços menores com o objetivo de mantê-los o mais independente possível. Para isso, cada serviço roda seu próprio processo e as comunicam são realizadas através de protocolos leves, como *RESTful*, no caso síncrono e *message queue*, no caso assíncrono.

2.2.1. APLICAÇÃO MONOLÍTICA

Para entender o padrão de microsserviços é importante entender o funcionamento do padrão monolítico (RAYMOND, E. 2003).

Uma aplicação monolítica é feita como uma única unidade. Ao se trabalhar com aplicações em três camadas, uma única aplicação de servidor irá manipular as requisições do usuário, executar toda a lógica de negócios, atualizar a base do banco de dados e enviar a resposta para o usuário. Qualquer mudança consiste em publicar uma nova versão dessa aplicação. Um esquema da diferença entre as arquiteturas está mostrado na Figura 4.

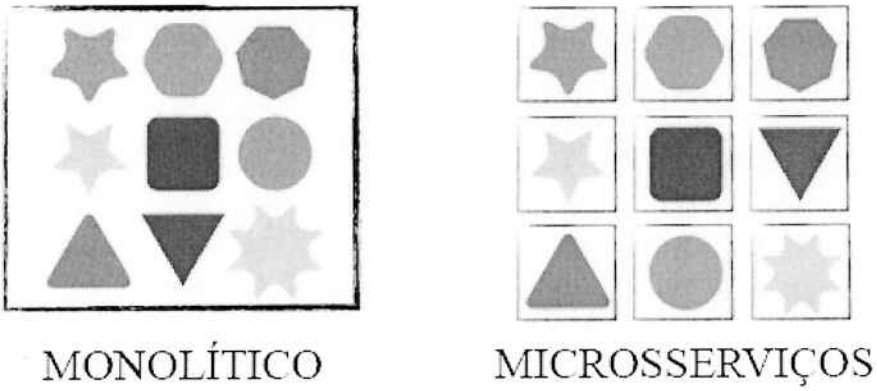

Figura 5 - Diferença entre monolítico e microsserviços

2.2.2. CARACTERÍSTICAS DA ARQUITETURA DE MICROSSERVIÇOS

2.2.2.1. Componentização via Serviços

Um componente é uma unidade de software que é substituída ou atualizada de maneira independente. Arquiteturas em microsserviços usam bibliotecas, mas também organizam seu próprio software dividindo em serviços. Bibliotecas são componentes que são usados em um programa através de chamadas diretamente em memória, e serviços são processos diferentes que se comunicam através de mecanismos como requisições remotas.

2.2.2.2. *Organização da Equipe através da Área de Negócios*

Ao dividir uma grande aplicação em partes, o foco é geralmente na camada de tecnologia, levando os times a serem divididos entre aqueles que cuidam da interface, dos serviços e do banco de dados. Ao dividir a equipe desta forma estamos aplicando a Lei de Conway, exemplificada na Figura 5:

"Qualquer organização que desenha um sistema irá produzir um design cuja estrutura é uma cópia da estrutura de comunicação da própria organização. " – CONWAY, M. 1967

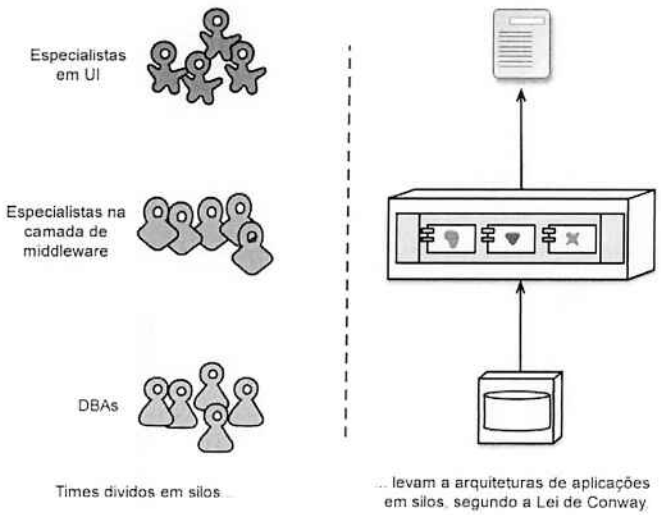

Figura 6 - Lei de Conway em ação (FOWLER, 2014)

A abordagem proposta pelos microsserviços para esta divisão é diferente, organizando os times ao redor das áreas de negócio. Assim, os serviços possuirão uma implementação completa para cada área do negócio, como a interface com o usuário, armazenamento de dados e aplicação *server-side*, como o esquema da Figura 6.

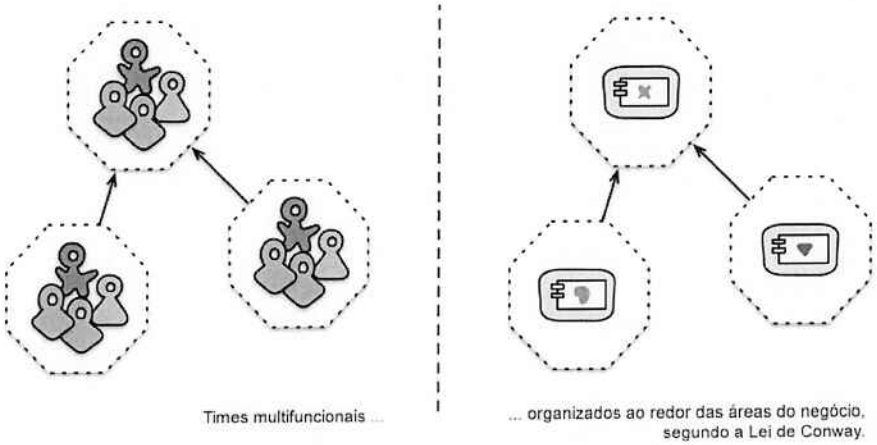

Figura 7 - Limites dos serviços reforçados pelos limites dos times (FOWLER, 2014)

2.2.2.3. Produtos e não Projetos

Em aplicações monolíticas é comum o modelo de projeto: onde o dever é entregar algum software que é por si só considerado o fim. Após isso, o software é entregue para quem cuidará da manutenção e o time que construiu o software é desfeito. Microsserviços evita esse modelo, dando preferência a ideia de que um time deve possuir o produto durante todo seu tempo de vida.

2.2.2.4. Endpoints Inteligentes e Fluxo de Comunicações Simples

Aplicações construídas a partir de microsserviços devem ser tão desacopladas e coesas quanto possível – recebendo uma requisição, aplicando a lógica apropriada e produzindo uma resposta. Isso pode ser gerenciado usando protocolos REST ao invés de protocolos complexos como WS-Choreography. Microsserviços devem usar princípios e protocolos que a *web* usa (IAN ROBINSON, 2013), como o HTTP.

Na estrutura monolítica os componentes são executados no mesmo processo através de invocação de métodos ou funções. Mudar uma estrutura monolítica para microsserviços é complicado nesse sentido porque varia o padrão de resposta.

2.2.2.5. Governança Descentralizada

Uma das consequências de governanças centralizadas é a tendência de padronizar tudo em uma única plataforma tecnológica. Essa abordagem é limitada por que para cada problema, a melhor solução pode requerer uma plataforma diferente.

Ao quebrar componentes monolíticos em serviços é possível construir cada um deles usando uma plataforma diferente. Isso otimiza o uso da linguagem escolhida, banco de dados e padrões de projeto.

2.2.2.6. Comparação de Microsserviços e SOA

SOA é uma abordagem de projeto onde múltiplos serviços colaboram e provem um conjunto de funcionalidades.

Os principais problemas relacionados ao SOA é ter protocolo de comunicação padrão pesado - SOAP, possuir um intermediário em seu middleware - podendo ser pago, e, principalmente, uma falta em sua definição de como os serviços são granulados. Não há uma forma aceita de como dividir um serviço grande em vários menores. Não se fala sobre o que é um serviço grande ou como tornar os serviços desacoplados.

2.2.2.7. Administração Descentralizada de Dados

Microsserviços proferem permitir que cada serviço gerencie sua própria base de dados, quer através de diferentes instâncias usando a mesma tecnologia de banco de dados, ou até mesmo usando diferentes sistemas de banco de dados – uma abordagem chamada Polyglot Persistence (FOWLER, M. 2011) exemplificado na Figura 7.

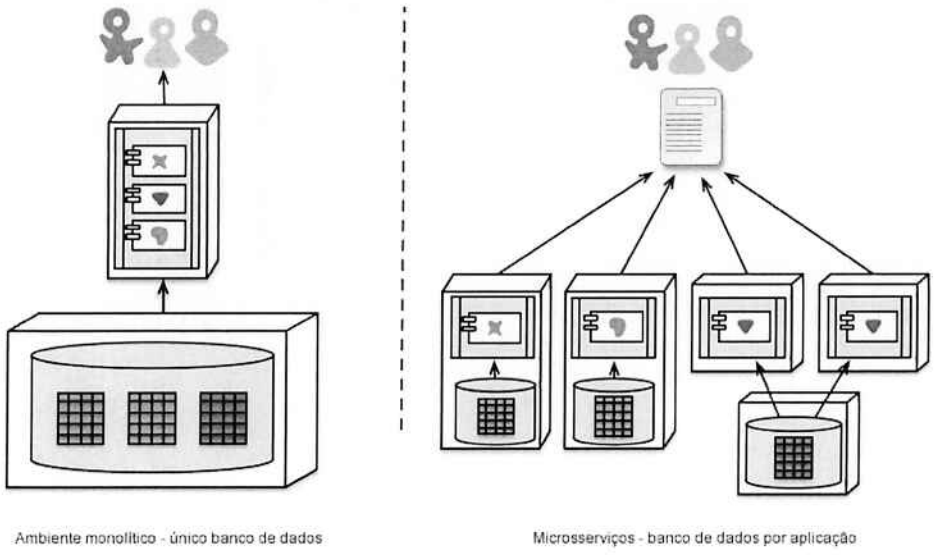

Figura 8 - Diferença entre abordagens de banco de dados (FOWLER, 2014)

A responsabilidade descentralizada para os dados através dos microsserviços tem implicações para atualizações. A abordagem comum usa transações, mas estas impõem um acoplamento temporário significante, que é problemático quando existem múltiplos serviços.

Os usos de transações distribuídas são nitidamente difíceis de implementar e por isso, arquiteturas de microsserviços enfatizam a coordenação sem transação entre serviços.

2.2.2.8. Projetado para Falha

Uma consequência do uso de serviços como componentes é que as aplicações precisam ser desenhadas de maneira que possam tolerar a falha dos serviços. Isso é uma desvantagem em relação ao monolítico porque introduz uma complexidade adicional ao desenvolvimento do *software*. Qualquer chamada de serviço poderá falhar devido à uma indisponibilidade do mesmo, e o cliente precisa responder a isso de forma esperada.

Microsserviços põem bastante ênfase no monitoramento em tempo real, checando elementos de arquitetura (por exemplo, requisições por segundo no banco de dados) e métricas relevantes ao negócio (por exemplo, quantos pedidos por minuto são recebidos). Tal monitoramento semântico pode prover um alerta antecipado de algo indo mal e levar os times de desenvolvimento a investigar o caso.

2.3. CLOUD COMPUTING

Computação em nuvem é uma computação que provê acesso a recursos como memória e armazenamento, por exemplo, sob demanda.

2.4. XCODE

Xcode é a IDE disponibilizado pela Apple para programar aplicativos para iOS e Macintosh. Atualmente, ela é a única plataforma para programar e compilar aplicativos para iOS e Macintosh, sendo obrigatório o seu uso para esses fins.

2.5. LIGHT TABLE

Light Table é um editor de texto que suporta as linguagens Clojure, Javascript e Python. Ele possui suporte ao NREPL, uma funcionalidade da Clojure que permite alterar o serviço rodando sem precisar o reiniciar, aumentando a produtividade de desenvolvimento.

Figura 9 - Light Table: Foto da tela de desenvolvimento

2.6. SWIFT 3

O Swift 3 é uma linguagem multi-paradigma de programação poderosa e intuitiva desenvolvida pela Apple para MacOS, iOS, watchOS e tvOS. Swift foi projetada para trabalhar com Cocoa and CocoaTouch – frameworks que auxiliam o desenvolvimento, e Objective-C – linguagem antiga de mesmo propósito do Swift. Swift teve seu código aberto em 23/11/2016. As principais vantagens dessa linguagem em relação ao antecessor Objetive-C são ser mais segura e concisa, facilitando o desenvolvimento.

2.7. ANGULARJS

AngularJS é um framework MVC de código aberto mantido pelo Google que auxilia na organização do código e em sua infraestrutura, muito usado quando a aplicação é dinâmica e complexa.

2.8. CLOJURE

Clojure é uma linguagem funcional derivada do Lisp, que combina a abordagem e o desenvolvimento iterativo de uma linguagem de *script* com eficiência e infraestrutura robusta

para programação *multi-thread*. Clojure compilada sobre a JVM, e provê fácil acesso às bibliotecas Java.

2.9. PEDESTAL

O pedestal é um framework de código aberto MVC para Clojure, com um conjunto de bibliotecas que traz para o desenvolvimento foco, poder e simplicidade para o desenvolvimento de servidor.

2.10. MONGODB

MongoDB é uma aplicação em código aberto de banco de dados NoSQL orientado à documentos, onde seus documentos tem a estrutura de um arquivo JSON.

2.11. REST

O RESTful ou REST é um estilo arquitetural que comporta seis princípios (REST, 2017):

- *Uniform Interface*: As interfaces entre os clientes e servidores devem ser as mesmas;
- *Stateless*: As informações necessárias para manipular as requisições estão nela própria, como parte da URI, cabeçalho ou corpo. O protocolo HTTP atende esse requisito muito bem;
- *Cacheable*: Os clientes podem deixar as respostas em cache. As respostas quem devem dizer se o cliente pode ou não a guardar em cache;
- *Client-Server*: O sistema deve seguir a arquitetura cliente-servidor padrão;
- *Layered System*: Os clientes não devem ter poder de decisão sobre enviar para o servidor de negócios ou um servidor intermediário, que aprimorar a escalabilidade através de cache e distribuidores de carga;
- *Code on Demand (opcional)*: O serviço pode estender a funcionalidade do cliente. Esse princípio é o único opcional e, caso outro princípio não seja implementado, então não se é considerado REST.

2.12. DOCKER

Docker é uma ferramenta projetada para facilitar a criação e hospedagem de aplicações através da utilização de containers. Containers permite ao desenvolvedor rodar uma aplicação gerada em pacote. Ao fazer isso, o desenvolvedor pode assegurar que a

aplicação funcionará em qualquer máquina, independente do sistema operacional e configurações locais.

Dessa forma, o Docker se parece com uma VM – *virtual machine*. Mas, ao invés de criar todo o sistema operacional, Docker permite à aplicação usar o mesmo Kernel do Linux. Isso aumenta o desempenho e reduz o tamanho da aplicação.

2.13. STRIPE

Stripe é uma plataforma que disponibiliza APIs para ajudar a criar a sessão de pagamentos de um sistema. O Stripe quem manipula as transações bancárias e armazena os cartões de crédito dos usuários que usam o sistema.

2.14. AMAZON WEB SERVICES

A AWS – Amazon Web Services – é uma suíte de serviços que oferecem as funcionalidades para implementar a *cloud computing*. Existem dezenas de serviços para se usar na AWS, contudo, nesse projeto foram usados os seguintes:

- *Elastic Compute Cloud – EC2*: Este serviço provê a possiblidade de criar e manipular instâncias de máquinas virtuais para diversos fins;
- *DynamoDB*: Este serviço provê a possibilidade de criar e manipular bancos de dados NoSQL;
- *EC2 Container Service – ECS*: provê a possibilidade de manipular e rodar containers do Docker em um *cluster* da AWS;
- *CloudWatch*: centraliza os logs dos containers da AWS ECS;
- *Route 53*: gerenciador de domínios e *buckets*;
- *Identify and Access Management – IAM*: Regula acessos das instâncias EC2;
- *Simple Storage Service – S3*: provê armazenamento de arquivos;

3. SISTEMA DE GERENCIAMENTO E AUTOMAÇÃO DE ESTACIONAMENTOS ORIENTADO À MICROSSERVIÇOS E CLOUD COMPUTING

3.1. METODOLOGIA DO PROJETO

Antes de apresentar a metodologia, é importante limitar o escopo deste projeto, de forma que tal escopo esteja dentro dos requisitos de um trabalho de conclusão de graduação do curso de engenharia de computação.

De forma geral, a metodologia consiste dos conceitos explanados pela engenharia de software padrão, aplicada no curso de graduação de Engenharia de Computação através da disciplina Laboratório de Engenharia de Software I, conforme livro-texto (BOOCH, G; RUMBAUGH, J.; JACOBSON, I. 2000) e (PAGE-JONES, M. 2001). A partir dos resultados das etapas da engenharia de software em questão, é separado os microsserviços e é aplicado os conceitos da arquitetura de software, este, assim como a engenharia de software, é lecionado no curso de graduação de Engenharia de Computação, através da disciplina Laboratório de Engenharia de Software II. Os microsserviços são separados como sugerido no livro de (FISHER, M. T.; ABBOTT M. L. 2009), onde, para escalar a aplicação através do eixo Y, devemos decompor a aplicação em serviços. A decomposição desse serviço, ainda com base no livro, se dá através da escolha de substantivos significativos do enunciado do projeto.

A partir do enunciado, os microsserviços são criados através da análise dos principais substantivos enunciados no projeto, os filtrando, levando em consideração a independência ou baixo-acoplamento entre os microsserviços. Com estes serviços, é feita uma análise com o intuito de descobrir o quão acoplado os microsserviços estão.

Com os microsserviços em questão, é criado uma versão inicial da arquitetura de três camadas dos casos de uso. Essa arquitetura tem, como objetivo, a estruturação e comunicação entre as camadas de apresentação, camadas de negócios e camadas de repositório.

Com posse dessa arquitetura é possível montar os Diagramas BPMN do sistema. O diagrama BPMN – Business Process Model Notation – é um modelo para representar como

estão estruturadas as lógicas de negócios do sistema. Nessa parte é solidificada a arquitetura de três camadas, a qual é modificada para uma versão final ao término dos diagramas BPMN.

Então, a finalização da arquitetura se dá esquematizando a arquitetura geral do sistema e especificando como está arquitetado cada serviço.

Primeiramente, a partir do enunciado do projeto, é descoberto quais são os atores e quais são os casos de usos que contemplam o escopo em questão, de forma a organizar e filtrar quais são as principais ações que serão executadas por cada ator. Com isso, é elaborado o Diagrama de Casos de Uso, onde é esquematizado como cada ator se comunica com o sistema. Esse Diagrama de Casos de Uso é parte primária do projeto e, a partir dele, se é elaborado a Descrição dos Diagramas de Casos de Uso, que descrevem os casos de uso de forma completa, o Diagrama de Classes e o Diagrama Interface Humano-Computador.

O Diagrama de Classes é elaborado a partir da junção do Diagrama de Casos de Uso com a metodologia do descobrimento de classes pela análise do enunciado do projeto e separação dos substantivos, com análise se faz sentido um determinado substantivo ser representado por uma classe.

O Diagrama Interface Humano-Computador, conhecido por IHC, é realizada através dos casos de uso de sistema estruturadas através de um fluxo, onde as telas que representam esse fluxo compõem o diagrama IHC.

A partir do Diagrama de Classes é possível criar o Diagrama de Sequências, que é representação dinâmica de como as classes, interfaces e entidades vão se comunicar em prol da realização de um caso de uso.

Com a conclusão do esquema acima, fica representável como o software completo deve se comportar.

3.2. DESAFIOS E FUNÇÕES DE PROJETO

O projeto piloto do **Parkaware** e que será abordado nesse trabalho é constituído de um sistema a ser implementado orientado à microsserviços que contenha interface gráfica *mobile* para o usuário (motorista) e plataforma web para os estacionamentos. De forma geral, o *software* a ser desenvolvido deve satisfazer algumas funções e se embasar em alguns desafios.

3.2.1. DESAFIOS

3.2.1.1. DESAFIO PARA USO DE MICROSSERVIÇOS

O nicho inicial do projeto está inserido no contexto de estacionamentos para *shopping centers*. Atualmente, o número de carros que vão à shopping centers no Brasil é de aproximadamente 100 milhões por mês. Isso resulta numa taxa média de 50 carros por segundo atravessando as cancelas do estacionamento e sendo capturado pelas câmeras do **Parkaware**. Diante disso, é necessário que nosso sistema seja capaz de escalar de forma que aceite dezenas de requisições por segundo. A princípio, o número de requisições é baixo, devido à porcentagem baixa que o software vai obter do mercado. Mas, conforme o sistema é instalado em outros shopping centers, é necessário que aguente o aumento de requisições sem perder desempenho.

Nesse sentido, a arquitetura de microsserviços supre as necessidades de escalabilidade que este mercado demanda.

3.2.1.2. DESAFIO PARA USO DE LINGUAGEM FUNCIONAL

Há anos, a Lei de Moore começou a falhar. Por mais que a parte da lei que prevê que o número de transistores em um processador dobra a cada dois anos ainda é obedecida, o desempenho do processador não está acompanhando o aumento do número de transistores de forma proporcional, como pode ser observado na Figura 10.

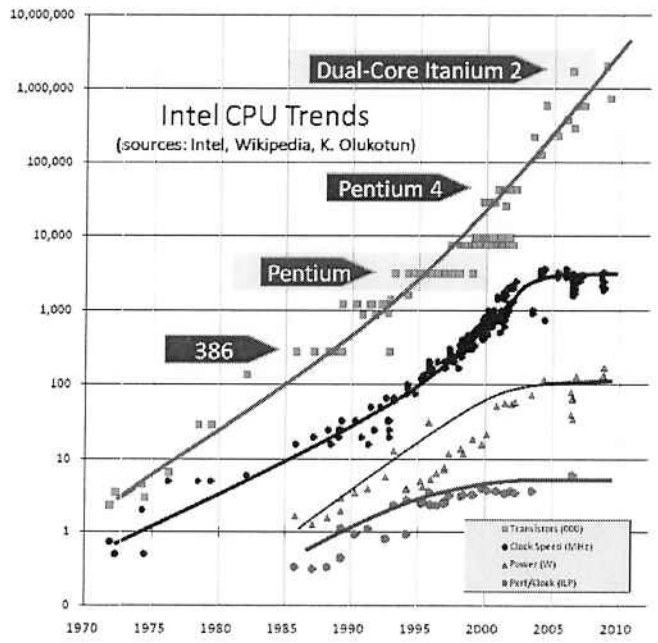

Figura 10 - Gráfico do número de transistores, frequência de clock, potência e desempenho/clock (SUTTER, H. 2009).

Para seguir a Lei, portanto, é necessária maior quantidade de processadores nos computadores, para que para o usuário aparenta que a Lei de Moore continua funcionando. O que deu origem aos processadores com vários núcleos.

Programar para processadores com vários núcleos é um desafio para o programador, uma vez que é complexo identificador qual thread está mudando uma variável do sistema.

Diante disso, o paradigma funcional (ABELSON, H.; SUSSMAN, G. J. 1979) emerge em um cenário onde se é necessária uma maneira de escrever código que não gerencia os estados das variáveis e, ademais, pudesse ser particionado para rodar em paralelo em quantos processadores forem necessários para o desenvolvedor. Deslocar um programa em linguagem funcional para trabalhar com concorrência é mais fácil do que se o programa estiver usando um outro paradigma de programação.

3.2.1.3. DESAFIO PARA USO DE CLOUD COMPUTING

A abordagem de microsserviços propicia a necessidade de deixar os serviços isolados em sua infraestrutura, levando em conta o princípio de Projetado para Falhar, conforma seção 3.1.2.8 desta monografia. Ademais, a possibilidade de criar mais serviços requer uma maior flexibilidade de infraestrutura.

A solução em *cloud computing* soluciona o desafio de ter toda a complexidade arquitetural de infraestrutura necessário para a abordagem de microsserviços a um baixo custo.

3.2.2. FUNÇÕES

Função	**Nome**	**Breve Descrição**
F0	Manipulação de Cadastro de Motoristas	O sistema permite o cadastro e gerenciamento de motorista (CRUD de motoristas)
F1	Reserva de Vagas	O motorista pode reservar uma vaga no estacionamento em determinado horário
F2	Gerenciamento de Entrada e Saída	O sistema deve ser capaz de saber reconhecer um motorista que entra e saí do estacionamento pela mensagem enviada do servidor local e mandar comandos de autorizando abertura ou fechamento da cancela para o servidor que enviou a requisição
F3	Disponibilidade de	O sistema deve ser capaz de calcular o número de vagas

	Vagas	disponíveis em cada estacionamento através de gerenciamento de entrada e saída de cada estacionamento distinto
F4	Pagamento automático	O sistema deve ser capaz de realizar o pagamento automático pelo cartão de crédito cadastrado de um motorista assim que é recebido uma mensagem de saída do seu carro
F5	Manipulação de Mensalistas	O sistema permite que motoristas de tornem mensalistas de um estacionamento, de forma a pagar um valor fixo e poder utilizar o estacionamento quantas vezes quiser por um mês
F6	Relatório de Estadias de Estacionamento	O sistema deve ser capaz de mostrar ao motorista um histórico detalhado de todas as vezes que usou o estacionamento
F7	Manipulação de Cadastro de Promoções	O sistema permite cadastro de promoções de lojas conveniadas por estacionamento
F8	Manipulação de Cadastro de Estacionamentos	O sistema permite cadastro e gerenciamento de estacionamentos (CRUD estacionamentos). A gerencia só pode ser feita pelo administrador do sistema
F9	Controle de Permanência	O sistema deve ser capaz de informar ao usuário em qual estacionamento está, o tempo de permanência em tempo real e o preço a ser pago pela estadia
F10	Emissão de Dados Estatísticos	O sistema monta estatísticas a partir dos dados registrados e seus usos, facilitando a análise do desempenho do negócio, como por exemplo: receita e disponibilidade por hora

3.3. ESPECIFICAÇÃO DE ARQUITETURA

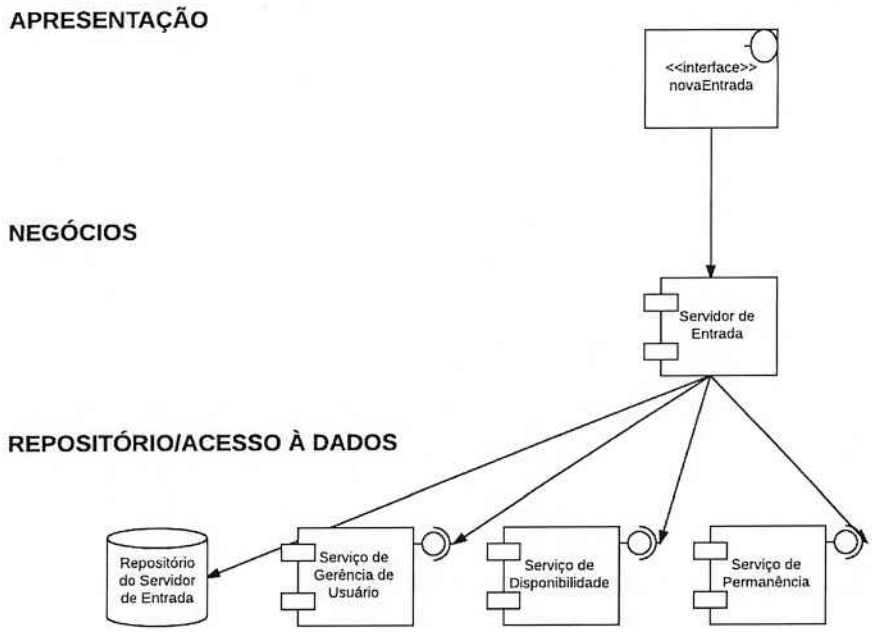

Figura 11 - Arquitetura de três camadas da entrada de um novo veículo

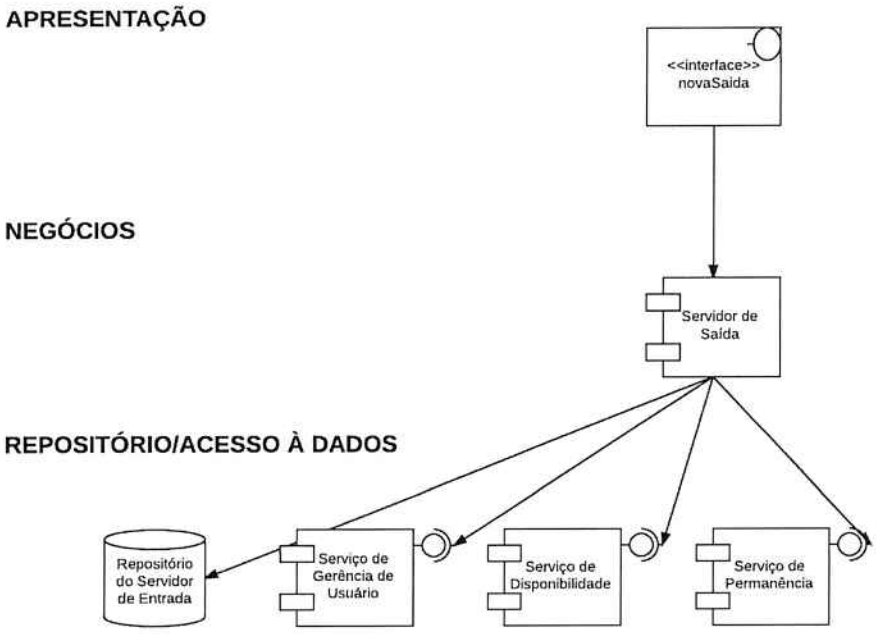

Figura 12 - Arquitetura de três camadas da saída de um novo veículo

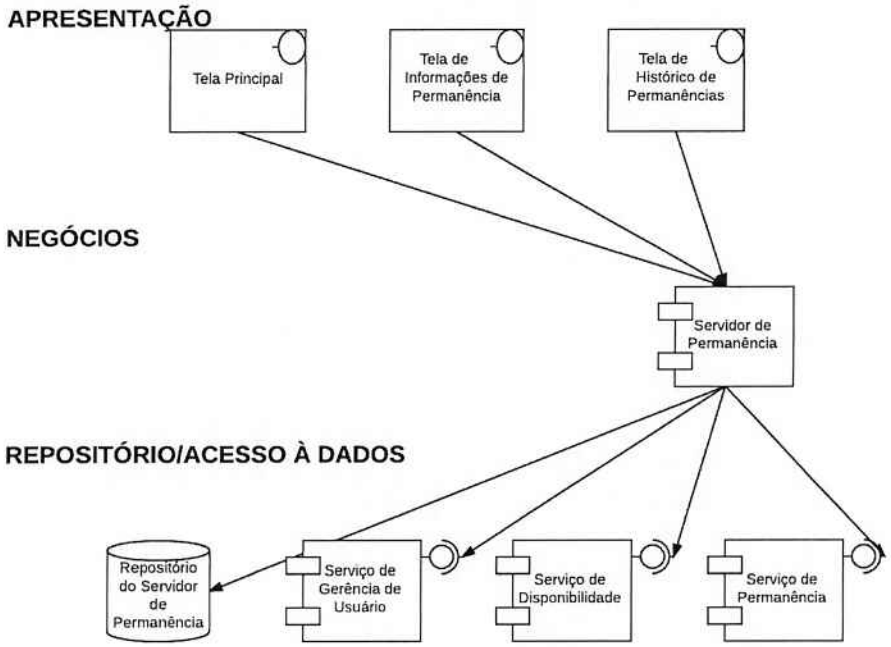

Figura 13 - Arquitetura de três camadas do tempo de estadia

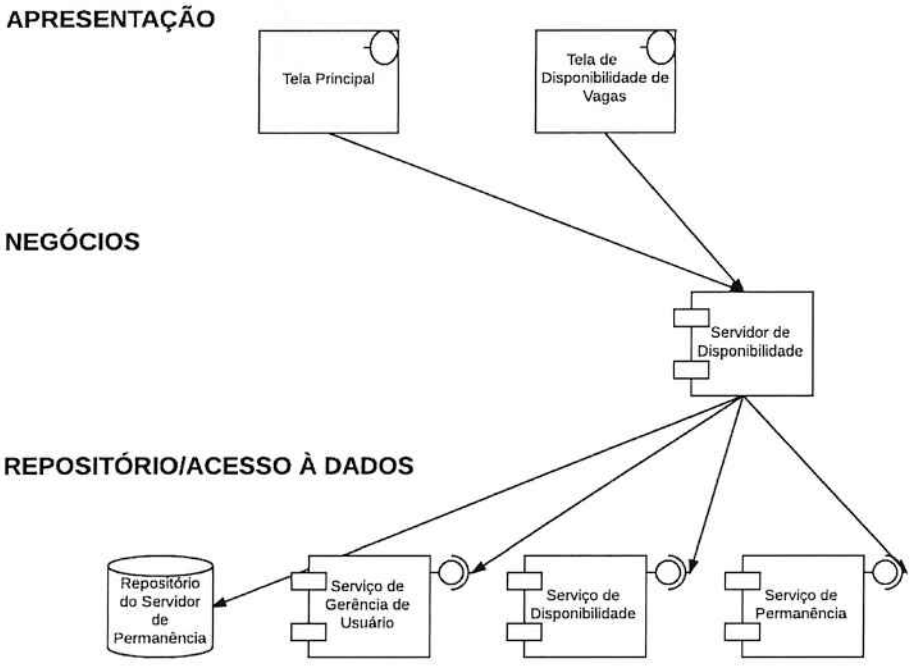

Figura 14 - Arquitetura de três camadas da disponibilidade

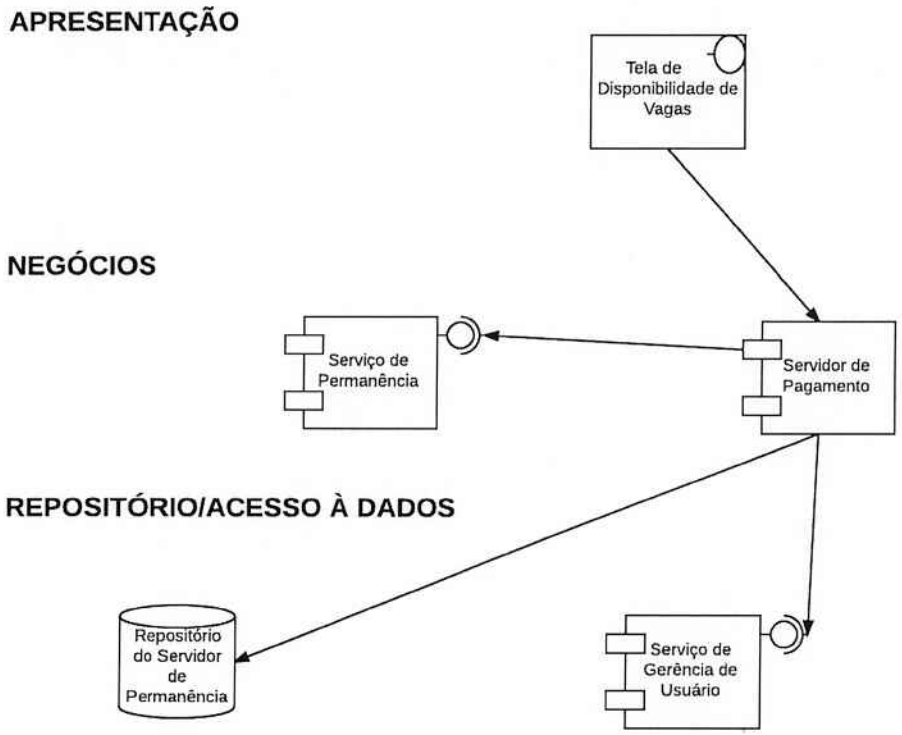

Figura 15 - Arquitetura de três camadas de fazer o pagamento

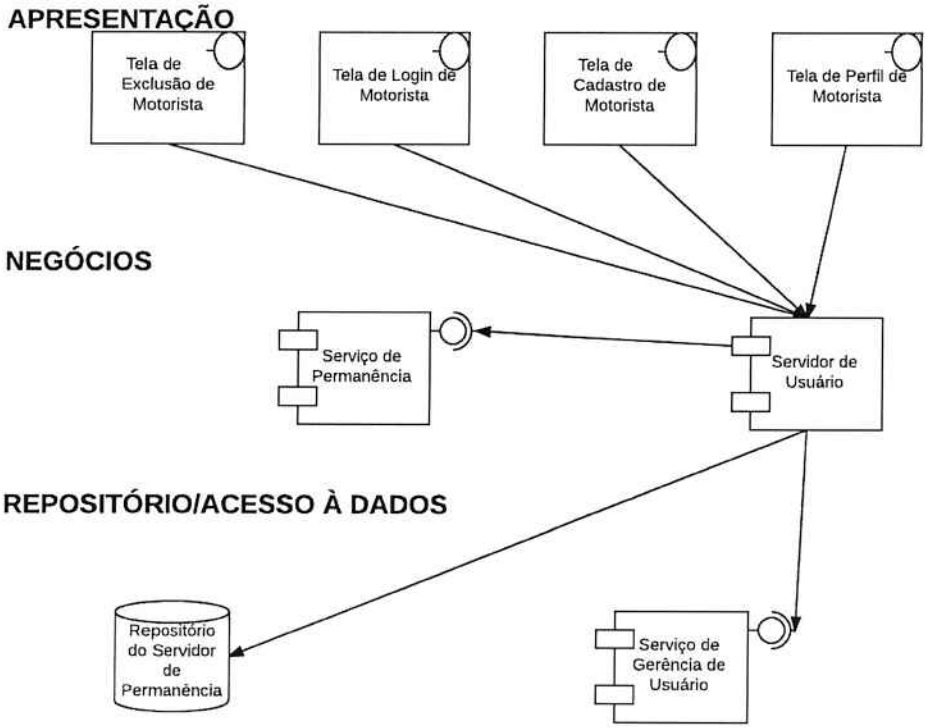

Figura 16 - Arquitetura de três camadas do CRUD

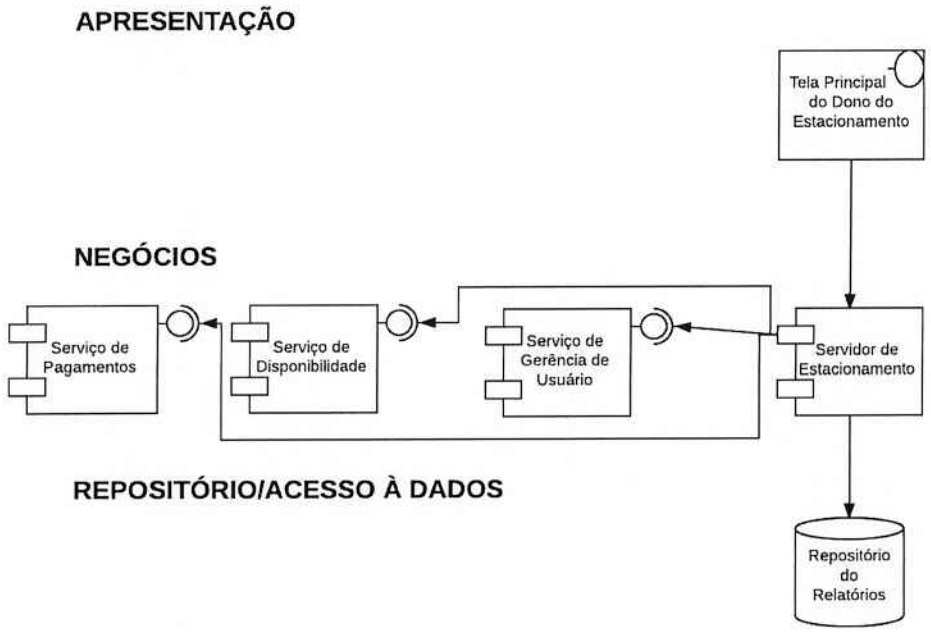

Figura 17 - Arquitetura de três camadas de dados estatísticos

3.3.1. MODELO DE ARQUITETURA DE SOFTWARE

Com as arquiteturas de três camadas da seção anterior, podemos construir a arquitetura geral de software do sistema. Esta arquitetura está disposta abaixo, na Figura 28.

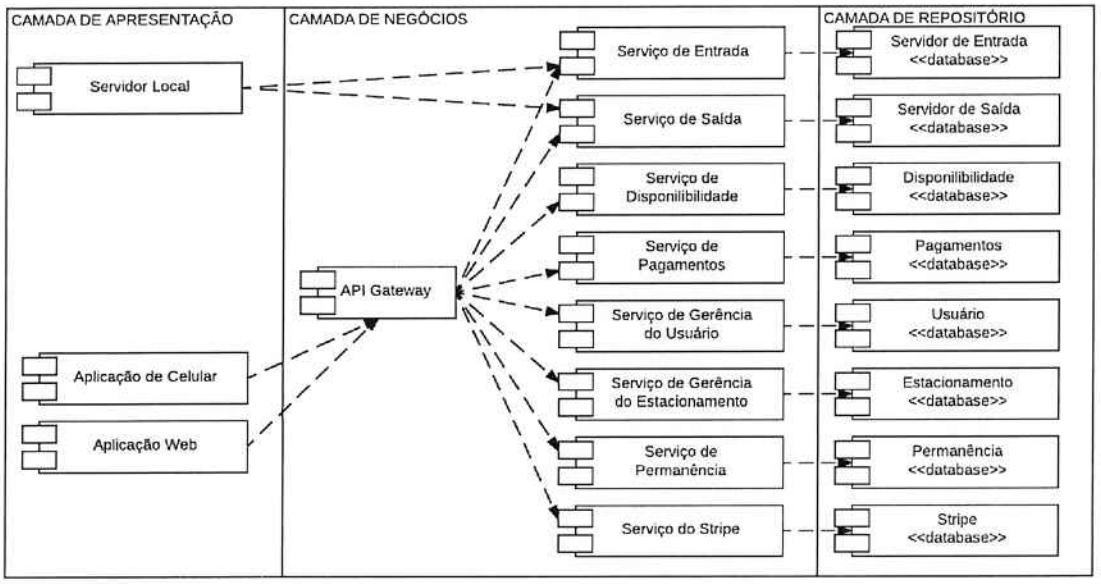

Figura 18 - Arquitetura de três camadas do sistema

3.3.2. MODELO DE ARQUITETURA DE SERVIÇO

Todos os microsserviços adotaram a mesma arquitetura, que está disposta abaixo na Figura 29.

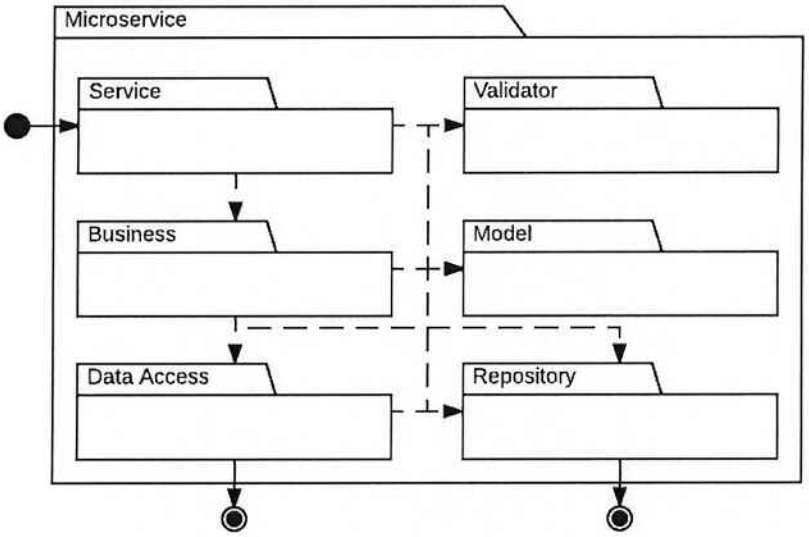

Figura 19 - Arquitetura de um serviço

3.3.3. MODELO DE BPMN

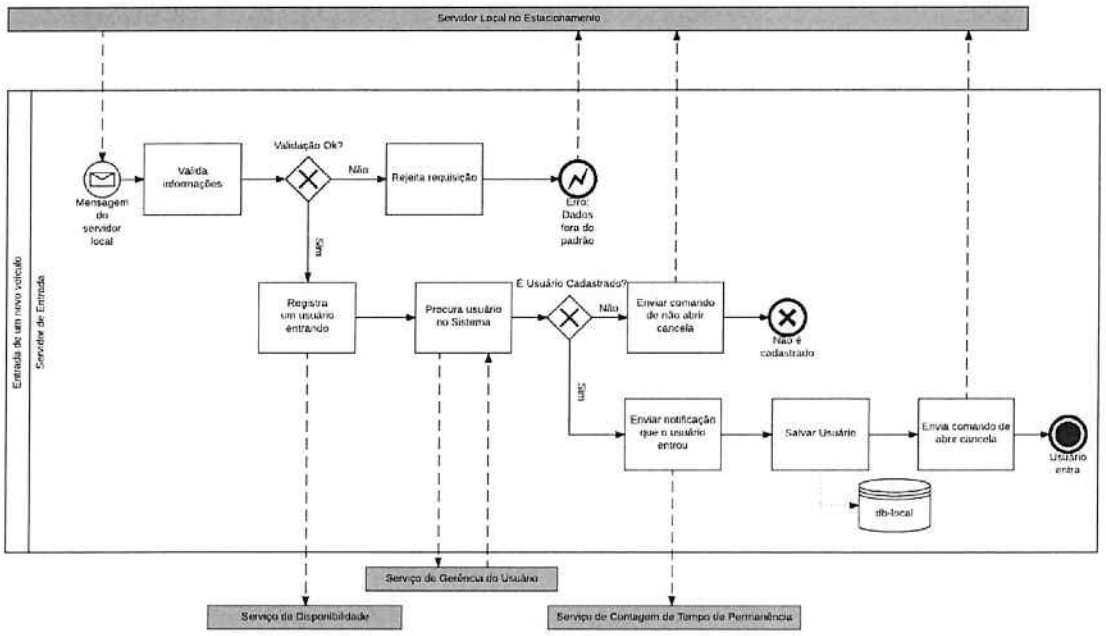

Figura 20 - BPMN De Entrada de um novo veículo

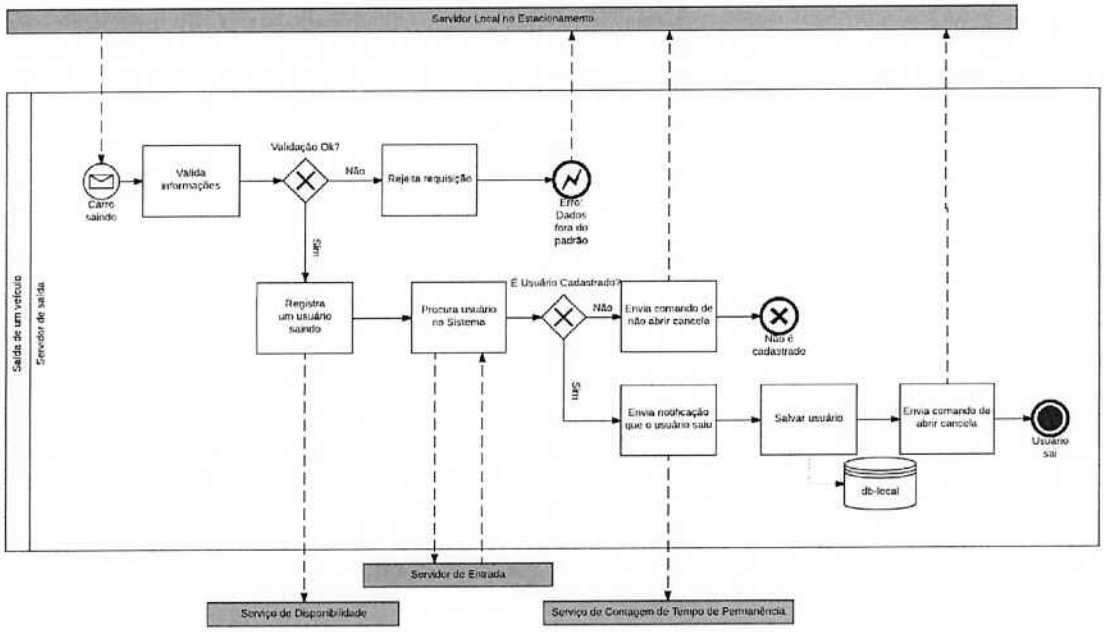

Figura 21 - BPMN de saída de um veículo

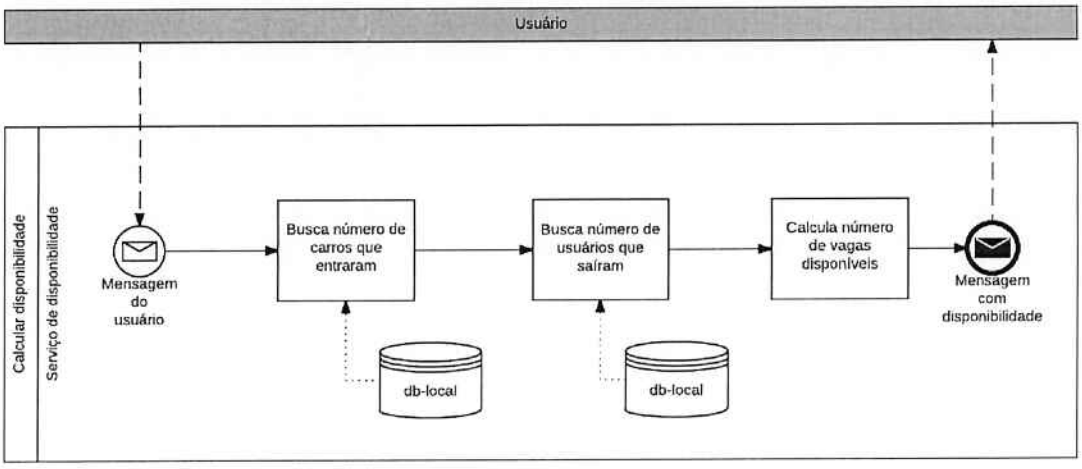

Figura 22 - BPMN de consultar disponibilidade

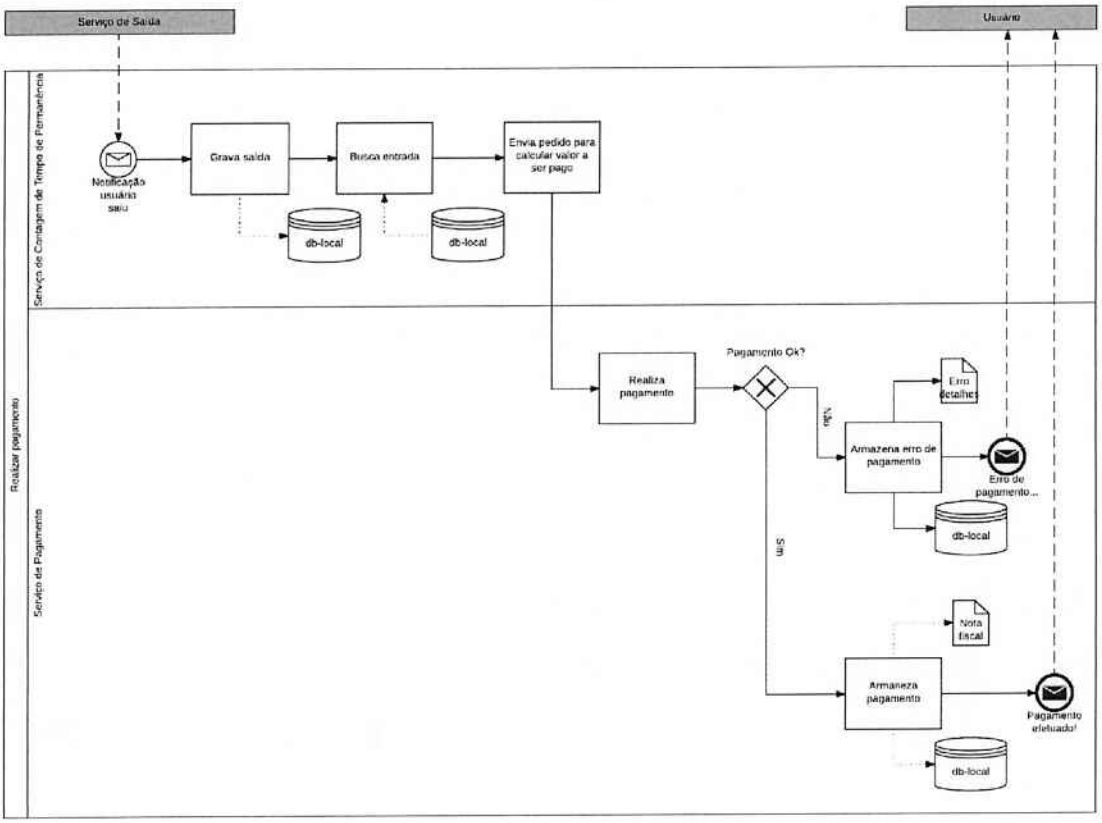

Figura 23 - BPMN de realizar pagamento

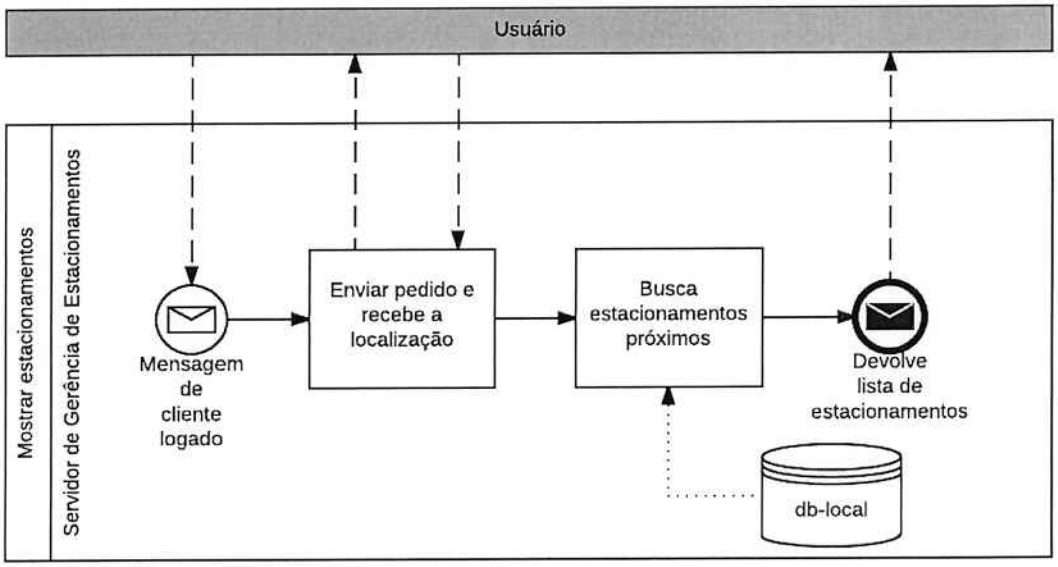

Figura 24 - BPMN de consultar estacionamentos

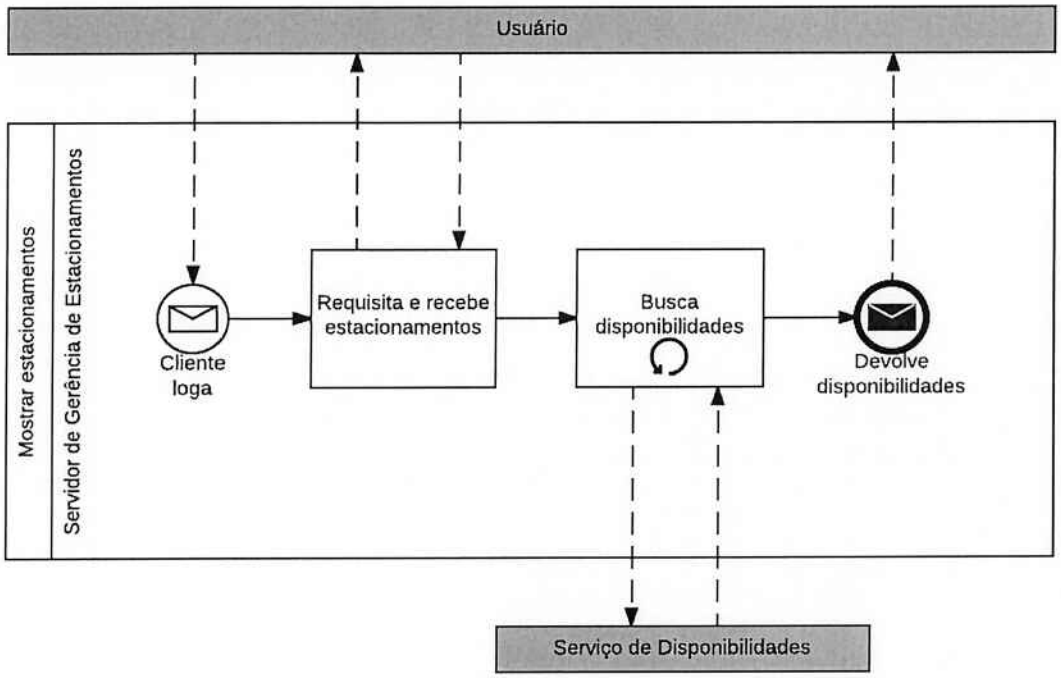

Figura 25 - BPMN de consultar disponibilidades

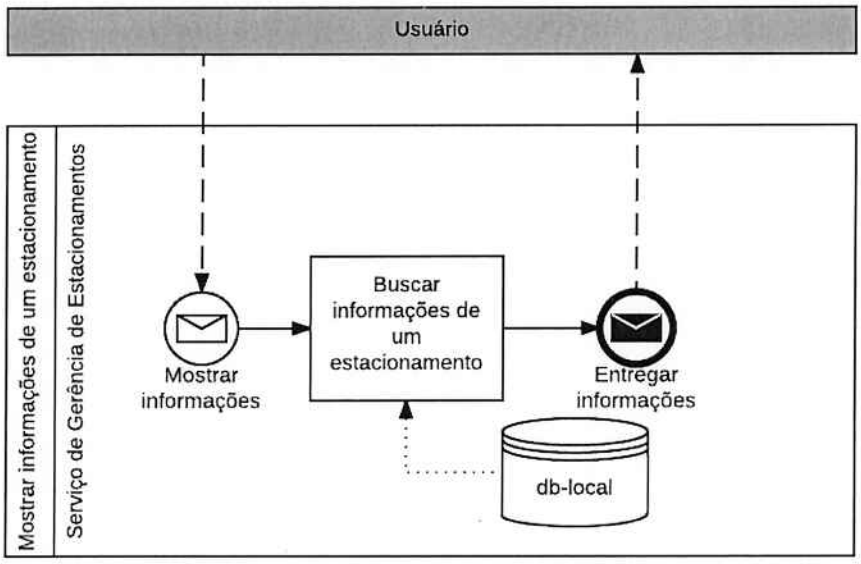

Figura 26 - consultar informações de um estacionamento

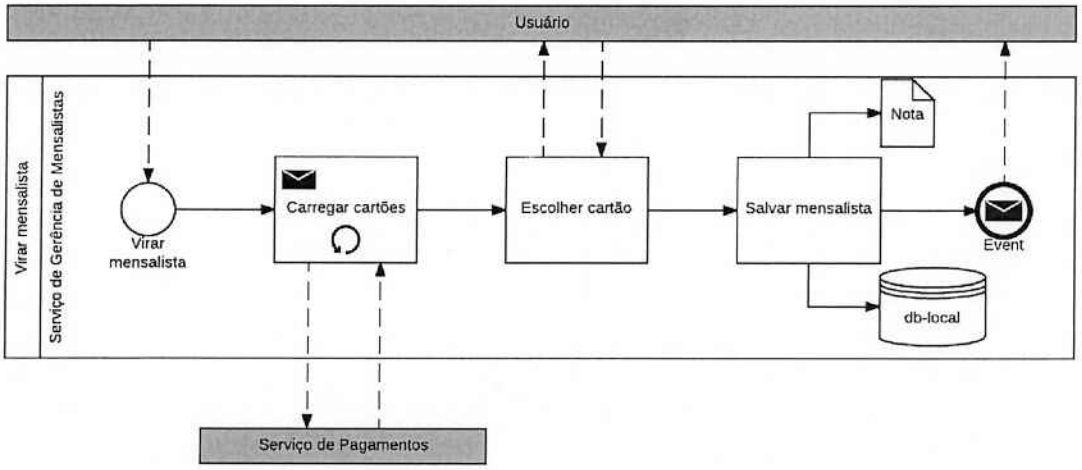

Figura 27 - cadastrar mensalista

3.4. ESPECIFICAÇÃO DE REQUISITOS

A especificação dos requisitos é composta pelo modelo estático e dinâmico, conforme a metodologia de engenharia de software. Para o modelo estático é elaborado o diagrama de casos de uso, diagrama de classes, diagrama de interface humano-computador. E, para o modelo dinâmico, é elaborado o diagrama de sequências. MODELO ESTÁTICO

3.4.1.1. Descrição dos Atores

Administrador - possui acesso a quase todas as funções do sistema. Sua função é resolver problemas técnicos e realizar modificações simples: por exemplo, cadastrar um estacionamento novo;

Motorista - é o condutor do veículo que deseja entrar no estacionamento usando o nosso serviço;

Loja - representa uma loja localizada dentro do estacionamento. Sua função no sistema é de prover promoções para os motoristas;

Sistema Local de Estacionamento - parte do sistema localizado dentro do estacionamento, que envia requisições para este sistema;

Dono do Estacionamento – responsável do estacionamento no nosso sistema.

3.4.1.2. *Diagrama de Casos de Uso*

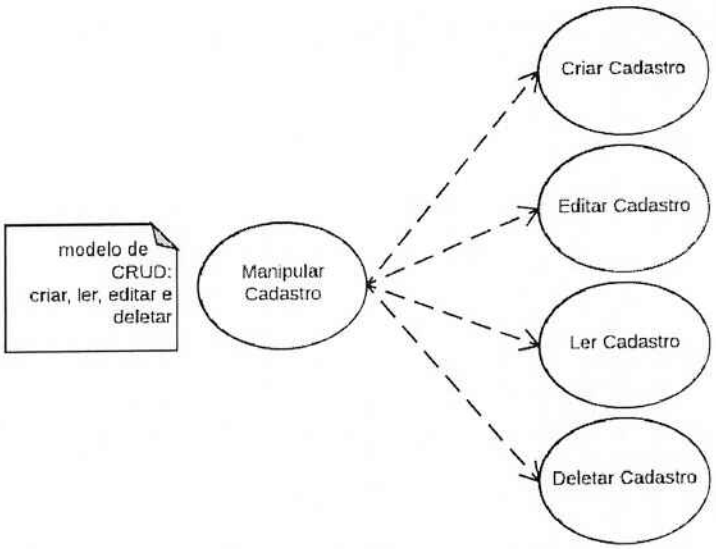

Figura 28 - Diagrama de Casos de Uso: Definição de Manipular

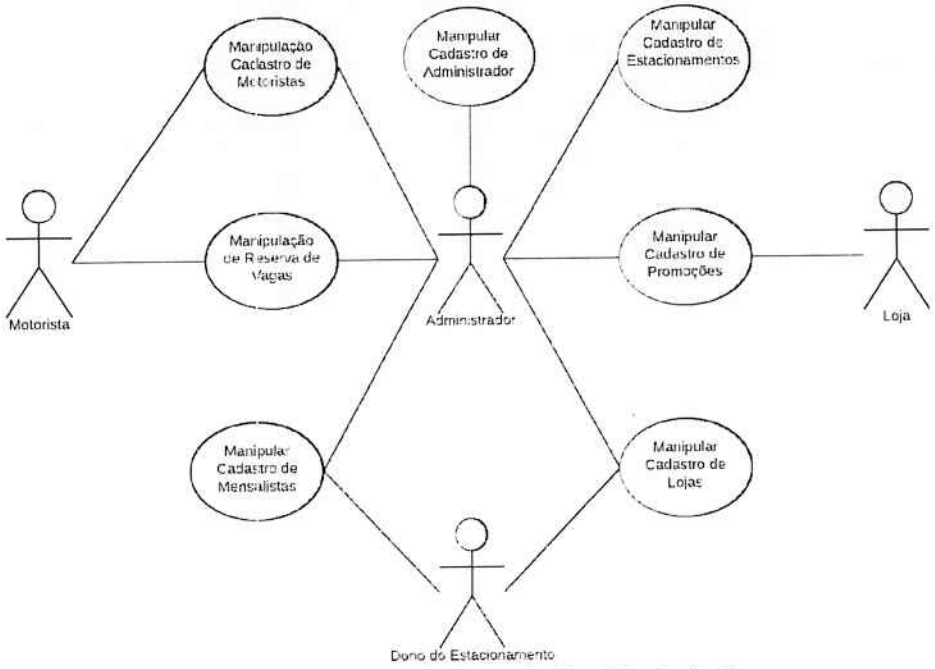

Figura 29 - Diagrama de Casos de Uso: Manipulações

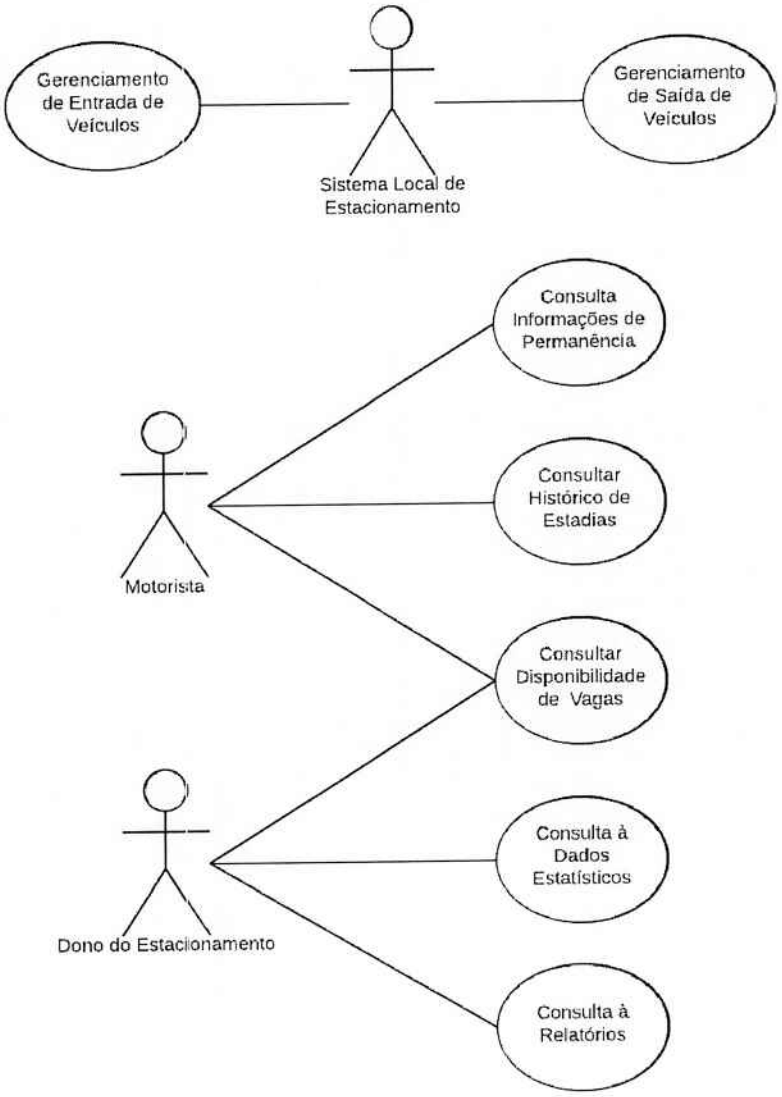

Figura 30 - Demais Diagramas de Casos de Uso

3.4.1.3. Descrição Sucinta dos Casos de Uso

Consultar informações de permanência - mostra ao ator informações de permanência da estadia do veículo no estacionamento, como: valor a ser pago e tempo de permanência;

Consultar histórico de estadias - mostra ao ator um histórico das vezes em que foi em estacionamentos e usou nossa plataforma, tendo acesso a informações de pagamento e tempo de permanência;

Consultar disponibilidade de vagas - mostra ao ator o número de vagas livres de um ou vários estacionamentos;

Consultar dados estatísticos - mostra ao ator dados gerados pelo sistema a partir de seus registros, facilitando decisões de negócio;

Gerenciar entrada de veículos - gerencia a entrada de um veículo, podendo ser um usuário cadastrado ou não. Para usuários cadastrados o sistema deve autorizar a abertura automática da cancela e; para não usuários, deve proibir;

Gerenciar saída de veículos - gerencia a saída de um veículo, podendo ser um usuário cadastrado ou não. Para usuários cadastrados o sistema deve autorizar a abertura automática da cancela e; para não usuários, deve proibir. Os pagamentos automáticos devem ser disparados por esse caso de uso;

Manipular cadastro de motoristas - manipula cadastro de motoristas;

Manipular cadastro de administradores - manipula cadastro de administradores;

Manipular cadastro de estacionamentos - manipula cadastro de estacionamentos;

Manipular cadastro de reserva de vagas - manipula cadastro de reserva de vagas;

Manipular cadastro de promoções - manipula cadastro de promoções;

Manipular cadastro de mensalistas - manipula cadastro de mensalistas;

Manipular cadastro de lojas - manipula cadastro de lojas.

3.4.1.4. Descrição Completa dos Casos de Uso

Diversos casos de uso correspondem às operações de criar, ler, atualizar e deletar, comumente denominadas como CRUD (do inglês, create, read, update e delete). Para evitar repetições, todas elas referenciarão o modelo a seguir e somente as características específicas serão detalhadas.

Caso de uso 0.1: Criar registro
Descrição: Cria o registro daquele objeto no sistema
Evento iniciador: Requisição de criação do registro
Atores:
Pré-Condição: Ator está autenticado no sistema.
Sequência de eventos:
Sistema solicita dados necessários para criação do registro.
Ator enviar os dados para o sistema.
Sistema verifica a consistência dos dados enviados
Sistema cria registro.
Sistema dá feedback sobre a criação do registro.

Sistema dá feedback sobre a criação do registro.

Pós-condição: Registro criado

Extensões:

Sistema apresenta os erros dos dados e solicita os dados novamente (Passo 3).

Inclusões:

Caso de uso 0.2: Visualizar registro

Descrição: Obtém as informações do registro daquele objeto no sistema

Evento iniciador: Requisição de leitura dos dados do registro

Atores:

Pré-Condição: Ator está autenticado e registro existe no sistema

Sequência de eventos:

Sistema busca o registro no banco de dados

Sistema apresenta, para o ator, o registro

Pós-condição: Registro apresentado na tela

Extensões:

Inclusões:

Caso de uso 0.3: Atualizar registro

Descrição: Modifica as informações o registro daquele objeto no sistema

Evento iniciador: Requisição de atualização dos dados do registro

Atores:

Pré-Condição: Ator autenticado no sistema. Registro deve ser editável.

Sequência de eventos:

Sistema solicita dados necessários para atualização do registro.

Ator enviar os dados para o sistema.

Sistema verifica a consistencia dos dados enviados

Sistema atualiza registro

Sistema dá feedback sobre a criação do registro.

Pós-condição: Registro atualizado

Extensões:

Sistema mostra erros de consistencia nos dados.

Inclusões:

Caso de uso 0.4: Remover registro

Descrição: Deleta o registro do sistema

Evento iniciador: Requisição de deletar o registro

Atores:

Pré-Condição: Ator autenticado no sistema e registro deve ser removível.

Sequência de eventos:

Sistema solicita a confirmação de remoção do registro

Ator confirma a remoção do registro

Sistema deleta o registro

Sistema dá feedback ao ator sobre a remoção do registro

Pós-condição: Registro removido

Extensões:

Ator cancela a remoção e o sistema permanece a tela atual (Passo 1)

Inclusões:

Demais modelos de caso de uso

Caso de uso 1: Consultar informações de permanência

Descrição: O ator acessa informações da permanência do veículo no estacionamento.

Evento iniciador: Ator clica em link que leva a tela de informações de permanência

Atores: Motorista

Pré-Condição: Ator está logado no sistema e dentro de um estacionamento

Sequência de eventos:

Sistema confirma autenticação de usuário

Sistema resgata horário de entrada do usuário em um estacionamento

Sistema calcula tempo de permanência

Sistema calcula valor a ser pago pelo usuário por estar usando o estacionamento durante aquele tempo

Sistema mostra na tela as informações de permanência.

Pós-condição: Informações de permanência do usuário é mostrada na tela.

Extensões:

O sistema também calcula o tempo para cobrar o valor do próximo período (passos 3 e 4).

Inclusões:

Caso de uso 2: Consultar histórico de estadias

Descrição: O ator acessa histórico das vezes que foi em estacionamentos usando o sistema.

Evento iniciador: Ator clica em link que leva a tela de histórico de estadias

Atores: Motorista

Pré-Condição: Ator está logado no sistema

Sequência de eventos:

Sistema confirma autenticação de usuário

Sistema resgata históricos por usuário

Sistema mostra na tela o histórico de estadia.

Pós-condição: Histórico do usuário é mostrado na tela.

Extensões:

Inclusões:

Caso de uso 3: Consultar disponibilidade de vagas

Descrição: O ator acessa informações da permanência do veículo no estacionamento.

Evento iniciador: Ator clica em link que leva a tela de informações de permanência

Atores: Motorista

Pré-Condição: Ator está logado no sistema e com GPS habilitado

Sequência de eventos:

Sistema confirma autenticação de usuário

Sistema busca dados de análise do estacionamento

Sistema calcula as distâncias do usuário até cada estacionamento

Sistema carrega as disponibilidades desses estacionamentos

Sistema mostra na tela os as informações de disponibilidade de cada estacionamento.

Pós-condição: Informações de disponibilidade dos estacionamentos é mostrado na tela do usuário.

Extensões:

O sistema também mostra a distância do usuário ao estacionamento (passos 3 e 5).

Inclusões:

Caso de uso 4: Consultar dados estatísticos

Descrição: O ator acessa dados estatísticos do estacionamento.

Evento iniciador: Ator clica em link que leva a tela de dados estatísticos

Atores: Dono do estacionamento

Pré-Condição: Ator está logado no sistema

Sequência de eventos:

Sistema confirma autenticação de usuário

Ator escolhe qual dado quer mostrar

Sistema faz cálculos e cria estatísticas sobre o dado

Sistema mostra na tela as estatísticas sobre o dado.

Pós-condição: Informações de estatísticas do dado são mostradas na tela.

Extensões:

O sistema também faz projeções em cima dos resultados (passo 3).

Inclusões:

Caso de uso 5: Gerenciar entrada de veículos

Descrição: O ator recebe informação de permissão sobre abrir a cancela para o motorista que está entrando.

Evento iniciador: Ator envia informações de entrada de um motorista

Atores: Servidor Local de Estacionamento

Pré-Condição: Servidor Local de Estacionamento conectado na internet e com posse do endereço de envio das informações do motorista

Sequência de eventos:

Sistema salva dados do motorista entrando

Sistema procura motorista pela placa

Sistema verifica que o motorista é cadastrado

Sistema inicia contagem de tempo de estadia do motorista

Sistema envia comando de permissão para abertura da cancela

Pós-condição: Motorista permitido para entrada.

Extensões:

Inclusões:

Se o motorista não for cadastrado (passo 3), não inicia contagem de tempo (passo 4) e envia comando de negação para abertura da cancela (passo 5).

Caso de uso 6: Gerenciar saída de veículos
Descrição: O ator recebe informação de permissão sobre abrir a cancela para o motorista que está saindo.
Evento iniciador: Ator envia informações de saída de um motorista
Atores: Servidor Local de Estacionamento
Pré-Condição: Servidor Local de Estacionamento conectado na internet e com posse do endereço de envio das informações do motorista
Sequência de eventos:
Sistema salva dados do motorista saindo
Sistema procura motorista pela placa
Sistema verifica que o motorista é cadastrado
Sistema termina contagem de tempo de estadia do motorista
Sistema envia comando de permissão para abertura da cancela
Sistema faz transação da conta do motorista para a conta da Parkaware
Pós-condição: Motorista permitido para saída e pagamento efetuado.
Extensões:
Inclusões:
Se o motorista não for cadastrado (passo 3), envia comando de negação para abertura da cancela (passo 5).

3.4.1.5. Diagrama de Classes

A seguir, encontra-se o Diagrama de Classes, representado pela Figura x1. Nela, contém os principais atributos e métodos que compõem as classes do sistema.

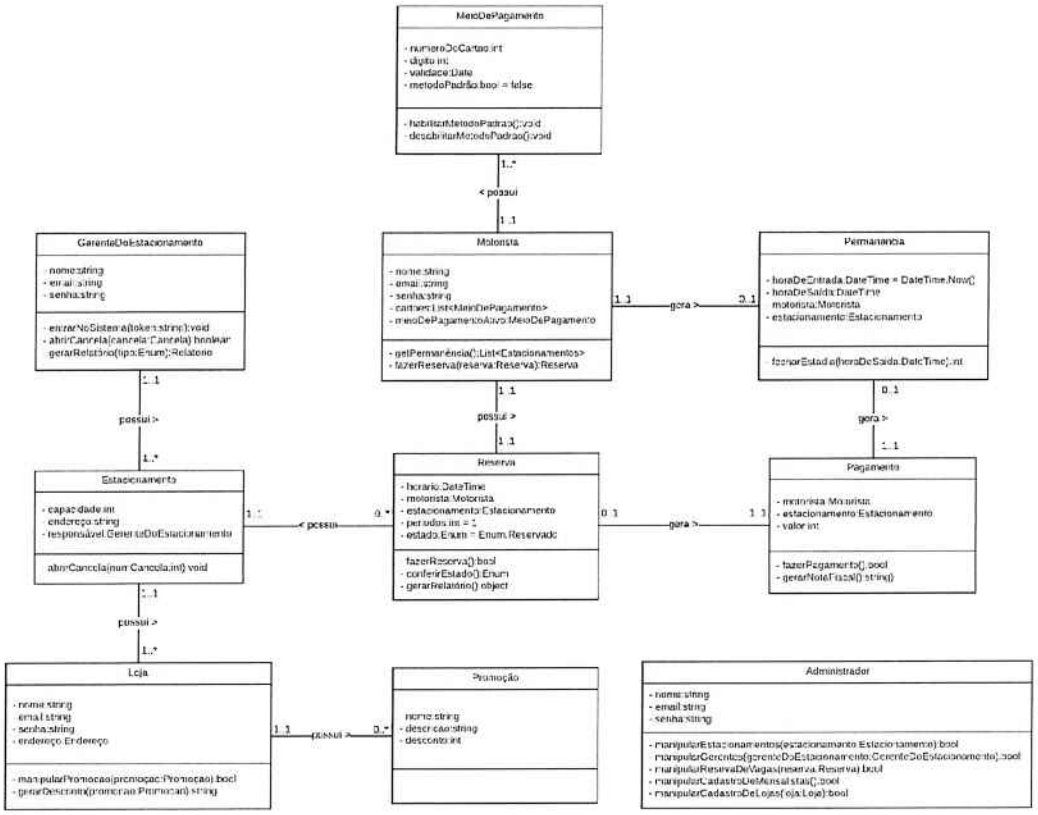

Figura 31 - Diagrama de Classes

3.4.1.6. Diagrama de Interface Humano-Computador

Abaixo estão representados os diagramas de interface humano-computador (IHC). Na Figura 15, temos os principais fluxos que podem ser realizados pelo motorista ao usar o aplicativo móvel. Na Figura 16, por sua vez, temos os principais fluxos que podem ser realizados pelo gerente ou dono ou responsável pelo estacionamento, o ator específico dependerá do estacionamento. Na Figura 17, temos o fluxo da loja, que poderá usar a plataforma nas próximas versões do sistema **Parkaware**.

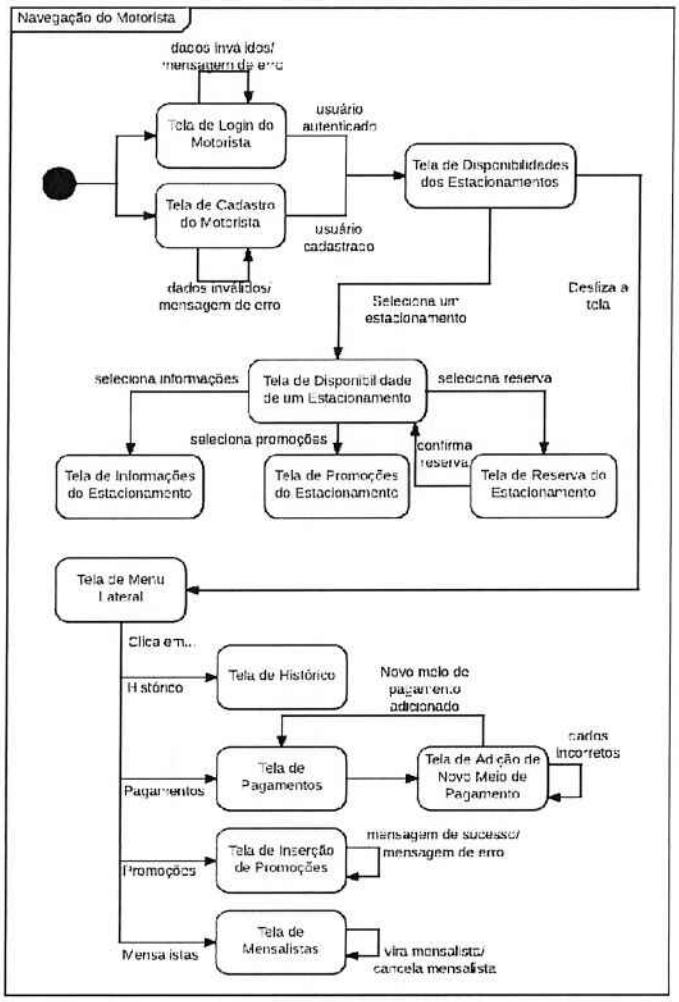

Figura 32 - Diagrama Interface Humano-Computador na visão do motorista

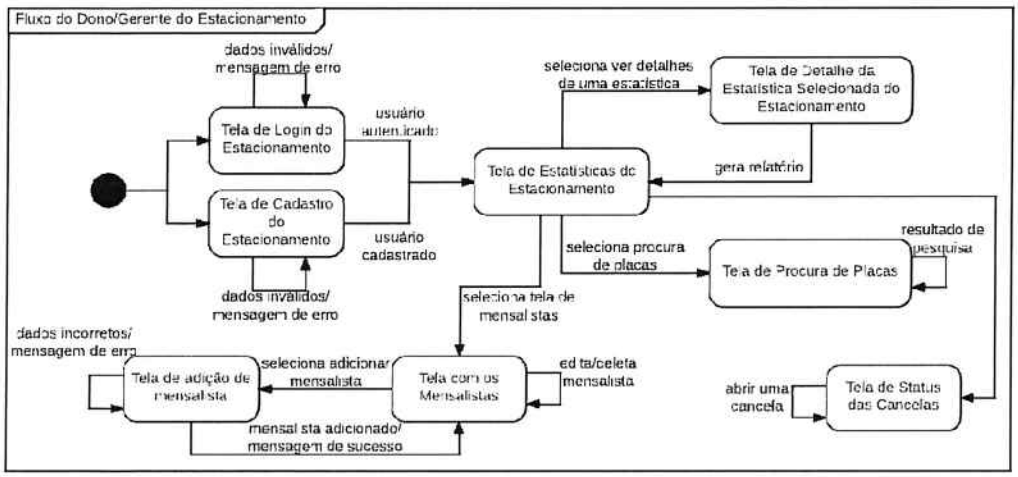

Figura 33 - Diagrama Interface Humano-Computador na visão do gerente do estacionamento

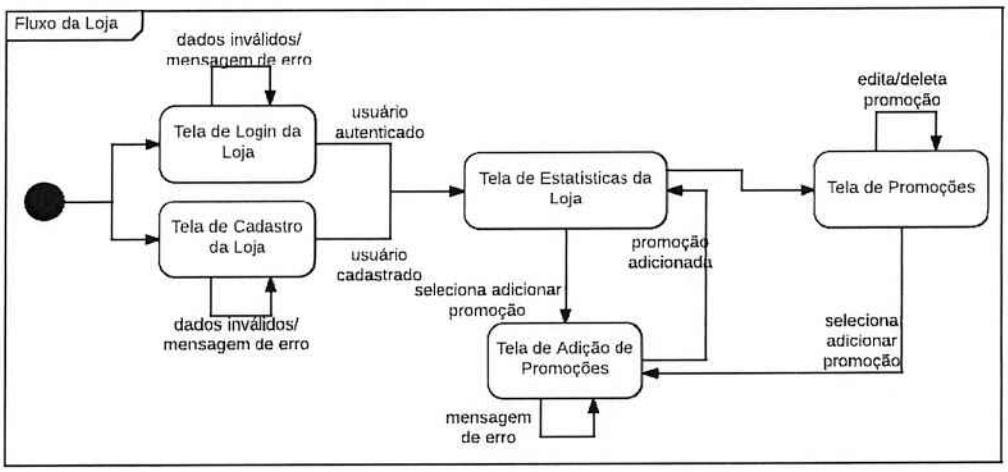

Figura 34 - Diagrama de IHC na visão da loja

3.4.2. MODELO DINÂMICO

Esta seção destina-se aos diagramas que compõem o modelo dinâmico da metodologia do sistema. Para tanto, foram selecionados os casos de uso mais complexos e com maior importância para serem colocados na monografia.

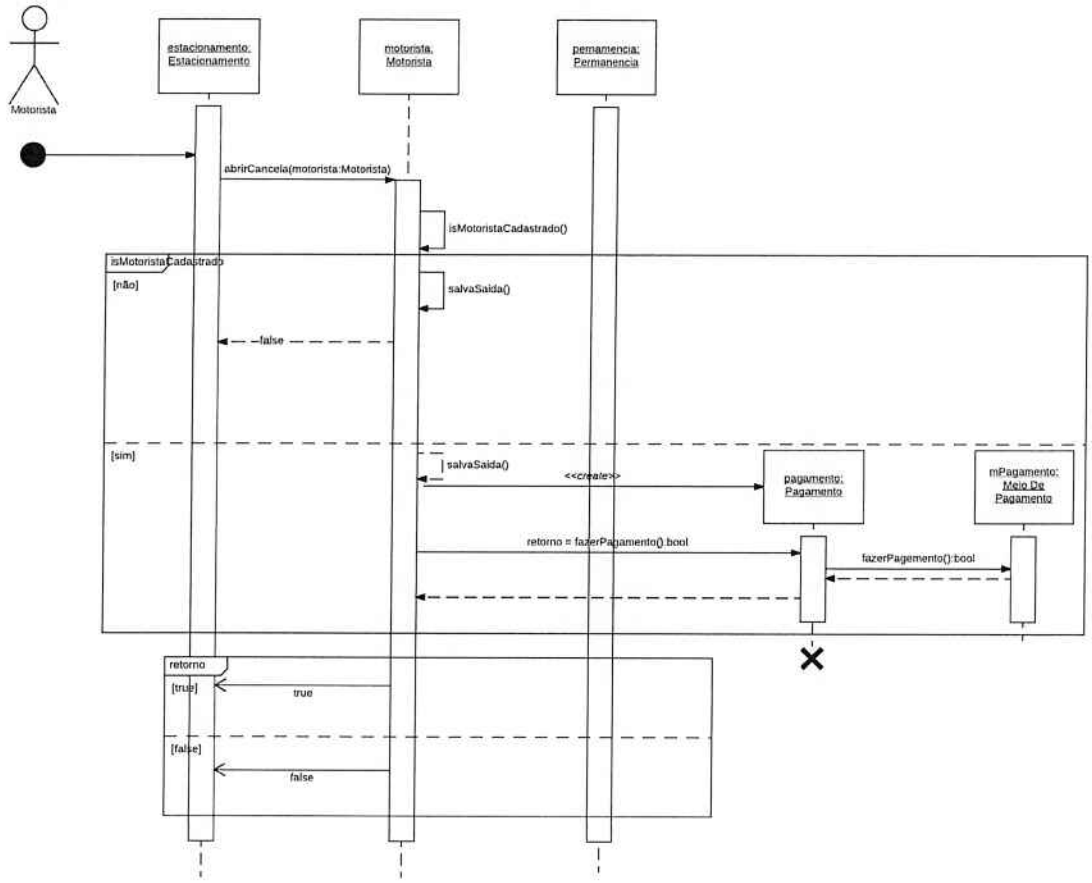

Figura 35 - Diagrama de Sequência de Gerenciar Saída do Estacionamento

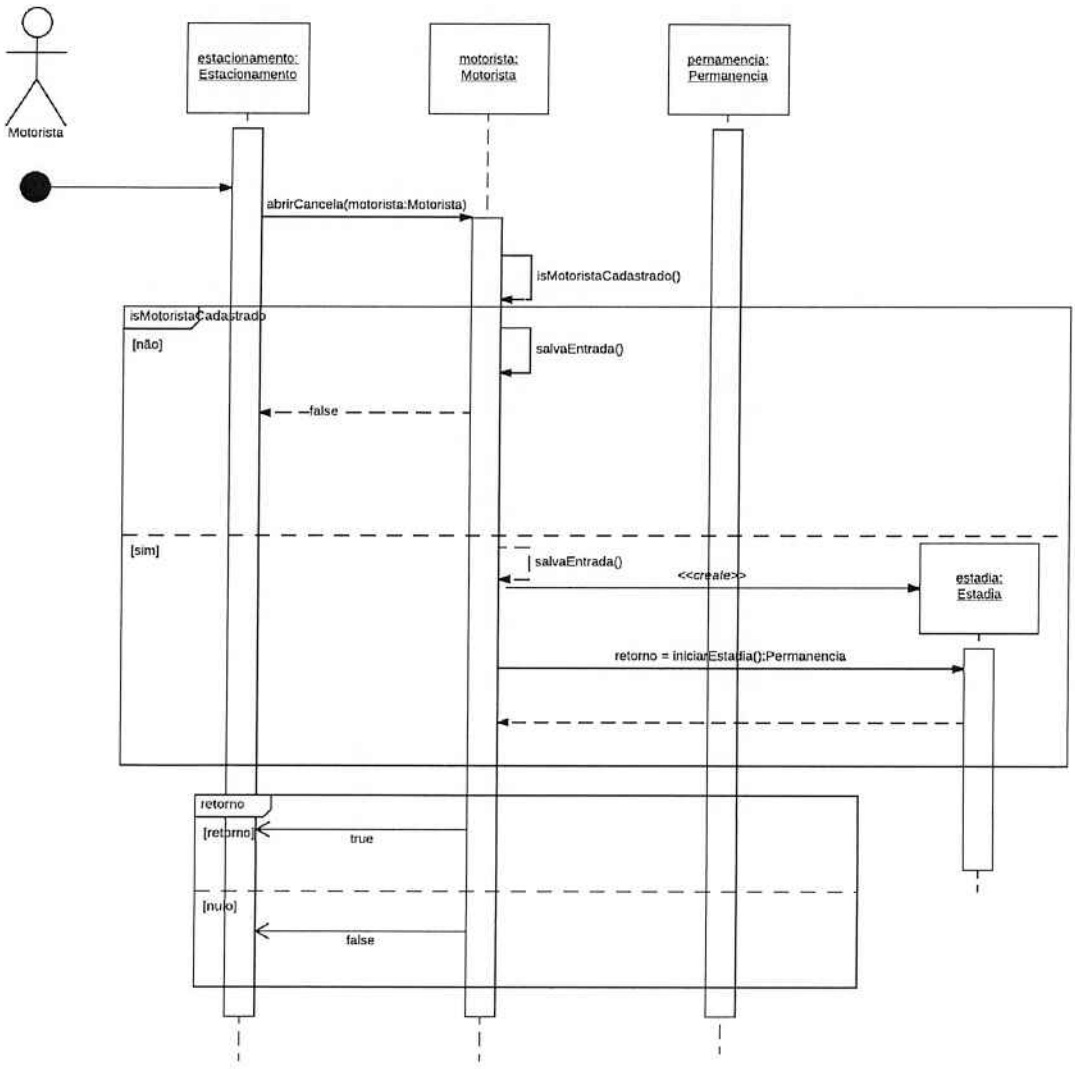

Figura 36 - Diagrama de Sequências de Gerenciar Entrada no Estacionamento

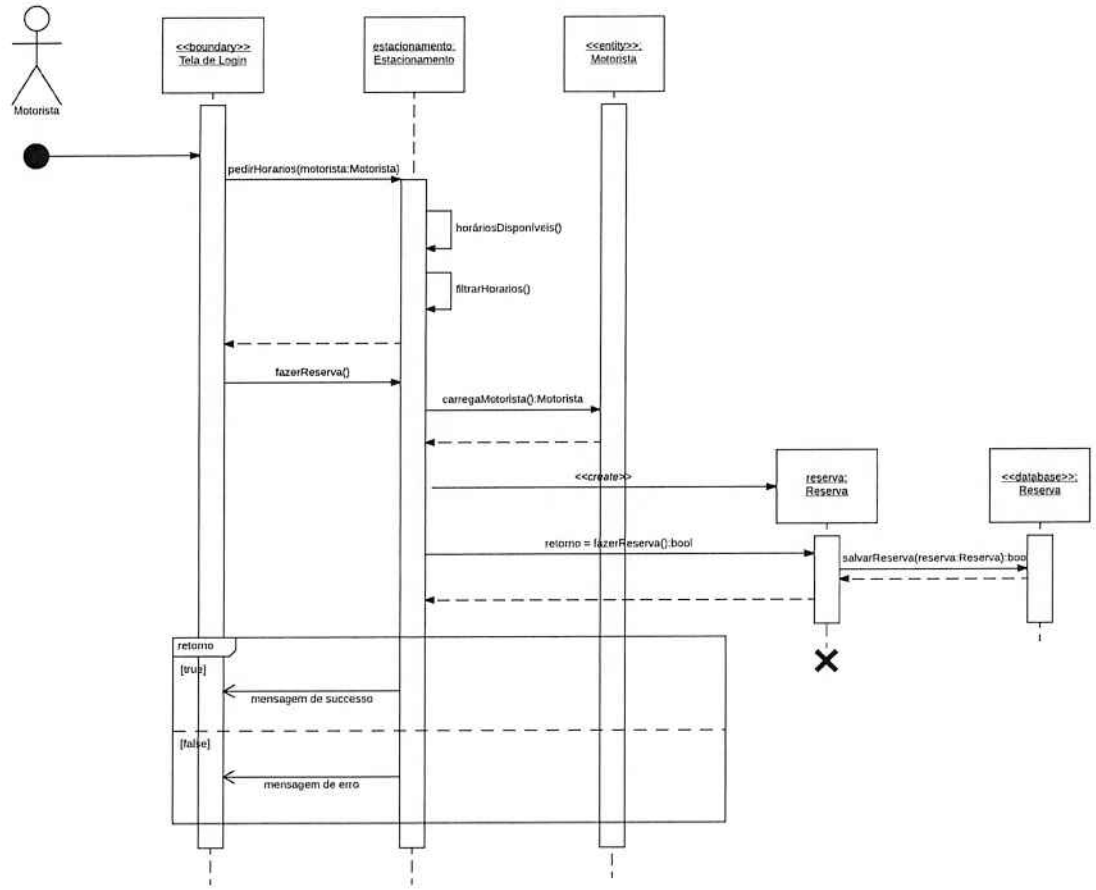

Figura 37 – Diagrama de Sequência de Fazer Reserva

3.5. IMPLEMENTAÇÃO

A implementação do sistema foi dividida em três partes: servidor, aplicativo móvel e plataforma web. As partes foram implementadas e testes foram realizados para garantir que o sistema atenda as funções de software especificadas anteriormente.

Neste capítulo são descritos separadamente os procedimentos executados para as três partes que compõem o sistema – servidor, aplicativo móvel e plataforma web. Inicialmente, é apresentado servidor separado em microsserviços e hospedado em *cloud*. Em seguida, é entraremos com maiores detalhes na aplicação móvel, explicando suas principais funcionalidades. E, para finalizar, explicaremos como foi implementado a plataforma web.

3.5.1. SERVIDOR

Para o desenvolvimento do servidor, utilizou-se a arquitetura de microsserviços. O *framework* escolhido para o esqueleto de um microsserviço foi o *Pedestal*. O *Pedestal* é um

framework que contém diversas bibliotecas que facilitam o desenvolvimento de uma aplicação de servidor.

Como na especificação, foram necessários criar nove microsserviços para mantê-los com o maior desacoplamento possível. Eles foram colocados dentro de um mesmo *cluster* da *Amazon Web Services* – AWS, conforme explicitado nas figuras X1 e X2. Um *cluster* é um agrupamento regional da AWS onde se pode rodar uma ou mais instâncias de containers.

Figura 38 - Dados do cluster da AWS onde foram publicados os serviços

Service Name	Status	Task Definiti...	Desired task...	Running task...
availability-service	ACTIVE	availability-tas...	1	1
api-gateway-service	ACTIVE	api-gateway-t...	1	1
stripe-service	ACTIVE	stripe-task-def...	1	1
entrance-service	ACTIVE	entrance-cont...	1	1
exit-service	ACTIVE	exit-task-defin...	1	1
payment-service	ACTIVE	payment-task-...	1	1
parking-lot-service	ACTIVE	parking-lot-tas...	1	1
user-service	ACTIVE	user-manage...	1	1
staying-service	ACTIVE	staying-task-d...	1	1

Figura 39 - Serviços publicados dentro do cluster

3.5.1.1. Comunicação

A comunicação se deu entre um serviço se comunicando com outros ou através de aplicações móvel e *web* para requisitar ou enviar informações. Se a requisição for enviada pelo aplicativo móvel ou *web*, então o serviço *api-gateway* recebe a requisição e se comunica com os outros serviços para obter a informações a ser respondida. Se a requisição vier pelo servidor local do estacionamento, então não há passagem pelo *api-gateway*, caindo direto nos serviços de entrada e saída, *entrance-service* e *exit-service*, respectivamente.

Essas comunicações foram realizadas de formas síncrona e assíncrona e, para cada tipo, um conceito e tecnologia diferente fora aplicado. Para a comunicação síncrona foi usado o padrão REST, com comunicações através de protocolo HTTP com operações de GET,

POST, UPDATE e DELETE. Para comunicações assíncronas foi usada uma plataforma de streaming com implementação a base de *message queue*, o ReactiveX.

3.5.1.2. Organização dos Arquivos

Todos os serviços seguem uma estrutura padrão de arquivos, como exemplo da Figura X1 está a estrutura do serviço de gerência de usuário, nomeado de *user-management*.

Figura 40 - Organização dentro de um serviço

No mais alto nível da estrutura existem três pastas principais: *config\, src\ e test* além dos arquivos *Dockerfile*, *project.clj* e *README.md*. Todas essas pastas e arquivos foram criadas junto com a criação do projeto pelo Pedestal.

Em *logback.xml*, localizado dentro de *config* é configurado o log do sistema com opções para tamanho do log, formato e diretório. Em todos os serviços a configuração padrão de 64Mb de tamanho do log, o formato é *{nome do serviço}-%d{yyyy-MM-dd}.%i.log* com diretório em *config\log* para salvar os logs.

Em *src* está o código principal do serviço. Em *business.clj* está lógica do serviço; *dbaccess.clj* tem toda a comunicação com o banco de dados; *repository.clj* possui todas as comunicações com os outros serviços; em *server.clj* se tem algumas funções de fábrica indispensáveis para o funcionamento do serviço; em *service.clj* estão todos os *endpoints* que esse serviço disponibiliza para os outros serviços ou aplicações; e, em *validator.clj* ficam todos os validadores de dados e negócios do serviço.

Em *test* ficam os testes do projeto. Em *service_test.clj* se testa se todos os endpoints estão funcionando de forma satisfatória.

Em *Dockerfile* ficam algumas configurações da imagem em que será criada a partir do serviço. Dados como porta que será usada e nome do .jar são configurados nesse arquivo.

Em *project.clj* é configurado todas as dependências desse serviço e quais bibliotecas o *Leiningen* precisa baixar porque serão usadas. O *Leiningen* é um gerenciador de dependências para Clojure.

Finalmente, o *README.md* é um arquivo que contém informações gerais do serviço como endereço dos *endpoints* e comandos para rodar a imagem.

3.5.1.3. Organização da Nuvem

Todos os serviços do projeto foram hospedados na nuvem através de uma suíte de serviços disponíveis pela AWS.

Primeiramente, foi criada uma imagem de cada serviço a partir das ferramentas do *Docker*. Então, foram realizados uploads dessas imagens para o repositório da AWS. A funcionalidade que gerencia repositórios de imagens de Docker está no serviço *EC2 Container Service*, ou ECS, da AWS que foi de onde se tirou a Figura 38 responsável pelos repositórios dos serviços.

Repository name	Repository URI	Created at
pa-entrance-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-09-19 14:32:33 -0300
pa-availability-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-17 00:00:12 -0200
pa-parking-lot-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-03 02:13:05 -0300
pa-stripe-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-16 13:28:20 -0200
pa-payment-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-17 02:27:59 -0200
pa-exit-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-17 02:51:52 -0200
pa-staying-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-03 02:11:29 -0300
pa-api-gateway-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-16 16:28:57 -0200
pa-user-service	389360187684.dkr.ecr.us-east-1.amaz...	2016-10-17 04:12:11 -0200

Figura 41 - Lista de repositórios

Essas imagens serão rodadas em containers do *Docker* que são configurados a partir da funcionalidade de *Task Definition* do ECS, mostrado na Figura 39. Em uma Task Definition é configurado a desempenho do container, memória, saída de logs, qual imagem

rodar, variáveis de ambiente, portas e outras configurações que não foram usadas neste projeto. Um exemplo dessa configuração está disposto na Figura 40.

Task Definition	Latest revision status
api-gateway-task-definition	ACTIVE
availability-task-definition	ACTIVE
entrance-container-task-definition	ACTIVE
exit-task-definition	ACTIVE
parking-lot-task-definition	ACTIVE
payment-task-definition	ACTIVE
staying-task-definition	ACTIVE
stripe-task-definition	ACTIVE
user-management-task-definition	ACTIVE

Figura 42 - Lista de *Task Definitions*

Container Definitions

Container Name	Image	CPU Units...	Hard/Soft...	Essential
pa-user-management-container	389360187684.dkr....	100	128/--	true

Details

Port Mappings

Host Port	Container Port	Protocol
49161	49161	tcp

Environment Variables

Key	Value
MONGO_CONNECTION	mongodb://admin:admin@ds040489.mlab.com:40489/testedb

Docker labels

Key	Value
No docker labels	

Extra hosts

Hostname	IP address
No host entries	

Mount Points

Container Path	Source Volume	Read only
No Mount Points		

Volumes from

Source Container	Read only
No volumes from	

Ulimits

Name	Soft limit	Hard limit
No ulimit		

Log Configuration

Log driver: awslogs

Key	Value
awslogs-group	parkaware
awslogs-region	us-east-1
awslogs-stream-prefix	user-service

Figura 43 - Configuração de uma *Task Definition*

Ao executar uma *Task Definition*, o container é criado e o serviço fica publicado e pronto para uso. Para garantir que o serviço continue rodando em caso de quebra, é usada a funcionalidade *Service* do ECS para que se assegure um número mínimo de cópias de containers que devem ser mantidas rodando ao mesmo tempo. No caso de ter mais que um container do mesmo tipo rodando, automaticamente é inserido um *Load Balancer* para distribuir e carga das requisições. Nesse projeto, os containers não foram duplicados e não houve necessidade de se trabalhar com *Load Balancer*.

Para finalizar com todos os serviços do projeto funcionando, coube utilizar o serviço IAM – *Identity and Access Management* – para configurar os usuários e IPs que terão acesso aos serviços do projeto.

3.5.2. APLICATIVO MÓVEL

O aplicativo móvel foi desenvolvido em Swift através da IDE *Xcode* da Apple. Este aplicativo é suporte para o motorista ter acesso às funcionalidades do projeto.

O Modelo de IHC da especificação tem um diagrama que trata da navegação que o motorista pode ter no aplicativo.

3.5.2.1. Bibliotecas Utilizadas

Foi necessário o uso da biblioteca *SwiftJSON* para converter os arquivos JSON das respostas das requisições para dicionários reconhecidos pelo Swift.

A manipulação HTTP nativa do Swift é difícil e complexa, então, foi necessário o uso da biblioteca *Alamofire* para trabalhar com as requisições HTTP de forma simplificada.

3.5.2.2. Organização dos Arquivos

A organização dos arquivos no projeto da aplicação móvel está disposta na Figura 41, onde se pode observar a estruturação de arquivos seguindo o modelo de MVC, onde Views e Controllers ficam localizadas no mesmo arquivos, ViewControllers.

Na pasta Model, temos as classes *AvailabilityCustomTableViewCell*, *ParkingLot*, *User*, *Vehicle*, que, respectivamente, representam uma célula da TableView de disponibilidade, o estacionamento, o motorista e o seu carro.

Na pasta ViewController, temos todas as classes ViewControllers, onde ficam a lógica de negócios e apresentação do aplicativo.

Em *AvailabilityTableViewController* está a lógica por trás da tela principal do sistema, representada pela Figura 42.a.

Em *LoginViewController* está a lógica por trás do carregamento do aplicativo e da tela de *login*, representada pela Figura 42.b.

Em PaymentViewController está a lógica por trás da seleção e inserção de cartões para o pagamento automático das estadias.

Na pasta Parkaware, temos os arquivos que compõem o fluxo do usuário através dos **.storyboard*, além de arquivos padrão do aplicativo.

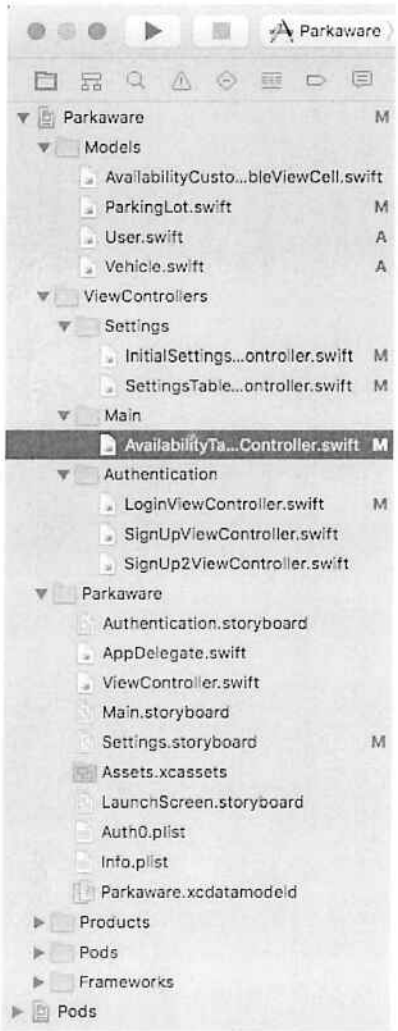

Figura 44 - Arquivos do aplicativo móvel

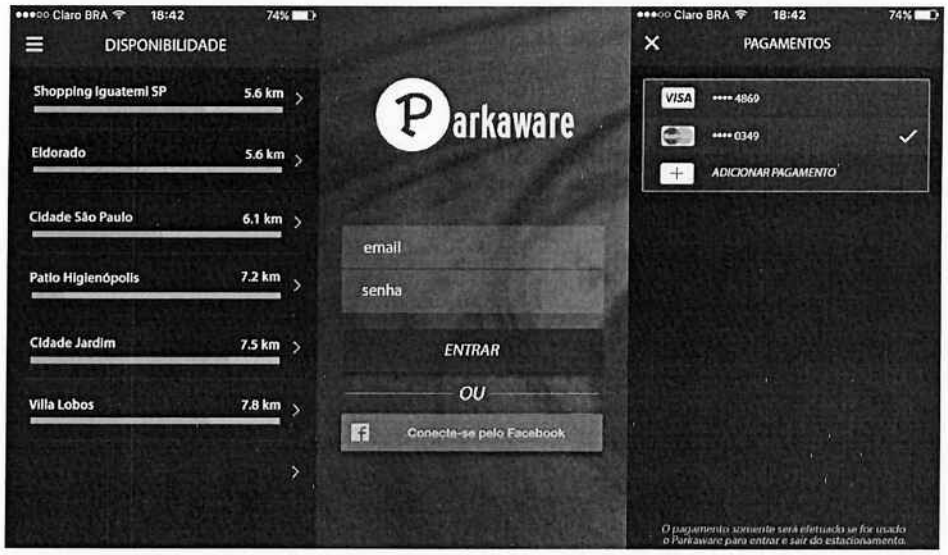

Figura 45 - a) Tela de disponibilidade. b) Tela de *login*. c) Tela de pagamentos

4. CONSIDERAÇÕES FINAIS

4.1. CONCLUSÕES

A arquitetura de microsserviços ajudou a escalabilidade do sistema quando possibilitou a divisão do servidor em vários menores, permitindo que somente os serviços que se tornam o gargalo do sistema sejam escalados, e não a aplicação completa, como sugere a arquitetura monolítica.

A metodologia usada baseada em arquitetura, ao focar a lógica do sistema, proporcionou uma visão geral de projeto que a metodologia de engenharia de software não supre por se embasar nas funções do sistema e participações dos atores.

O uso de *cloud computing* se mostrou complexo no contato inicial, mas, com o avançar do projeto, se fez eficaz ao facilitar o uso de diversos serviços sem ter que se preocupar com pormenores de infraestrutura, podendo dar maior atenção aos detalhes de desenvolvimento. E, juntamente com a arquitetura de microsserviços, se fez útil ao facilitar a escalabilidade pela replicação de serviços. O uso de *cloud computing* está intimamente ligado ao uso de microsserviços, por ser, atualmente, a melhor forma de separar os serviços em diferentes máquinas e escalonar de forma facilitada sem grandes custos.

O uso de linguagem funcional se mostrou compensador pela abstração que possibilitou partes que puderam ser replicadas entre os serviços. O uso de *multithreads* se fez de forma fácil, sem precisar se preocupar com múltiplos acessos de variáveis e com soluções simples que resolvem problemas de concorrência.

A monografia e a prática deste trabalho pretendem deixar uma contribuição para os futuros projetos que usarão arquitetura de microsserviços em Clojure e *cloud computing*.

Atualmente, a **Parkaware** está desenvolvendo um projeto que não existe no mercado e que facilitará a vida de muitos frequentadores de shopping centers, que não terão preocupações em relação ao carro e poderão se dedicar a outras atividades propiciadas pelo shopping.

O desenvolvimento do trabalho agregou para o integrante conhecimentos sobre linguagens novas, o paradigma da linguagem funcional, o conceito de arquitetura de

microsserviços, o *conceito de cloud computing, além de alertar sobre o envolvimento da literatura acadêmica* sobre os tópicos que estão em pauta no mercado.

4.2. PRÓXIMOS DESAFIOS

Para dar continuidade no projeto **Parkaware**, devem ser enfrentados uma série de desafios que estão dispostos nessa seção.

Estimar o número de carros que estarão em um determinado dia no estacionamento a partir de análise estatística e por Big Data Analytics, de forma a otimizar o recurso de reserva de vagas, onde, para cada instante de tempo, o número de vagas que podem ser reservadas é variável conforme a análise.

Além disso, implementar o mapeamento de vagas por câmeras e guiar o motorista até uma vaga por meio de inteligência artificial.

REFERÊNCIAS BIBLIOGRÁFICAS

ABELSON, H.; SUSSMAN, G. J. **Structure and Interpretation of Computer Programs**, MIT Press, 1979.

BOOCH, G; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário**, Editora Campus, 2000.

CONWAY, M. **How Do Committees Invent?** Datamation, 1967.

DAYA, S.; DUY, N. D.; EATI, K.; FERREIRA, C. M.; GLOZIC, D.; GUCIR, V.; GUPTA, M.; JOSHI, S.; LAMPKIN, V.; MARTINS M.; NARAIN, S.; VENNAN, R. **Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach**, Redbooks, 2015.

FISHER, M. T.; ABBOTT M. L. **The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise**, Addison Wesley, 2009.

FOWLER, M. **Polyglot Persistence**, 2011 Disponível em <http://martinfowler.com/bliki/PolyglotPersistence.html>. Acesso em 14 nov.2016.

FOWLER, M. **Microservices**, 2014 Disponível em <http://www.martinfowler.com/articles/microservices.html>. Acesso em 13 nov.2016.

HIGGINBOTHAM, D. **Clojure for The Brave and True: Learn the Ultimate Language and Become a Better Programmer**, No Starch Press, 2015.

KIM, G.; BEHR, K.; SPAFFORD, G. **The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win,** It Revolution Press, 2013.

NEWMAN, S. **Building Microservices: Designing Fine-Grained Systems**, O'Reilly, 2015.

PAGE-JONES, M. **Fundamentos do Desenho Orientado a Objeto com UML**, Makron Books, 2001.

PALA, Z.; INANC, N. **Smart Parking Applications Using RFID Technology**. RFID Eurasia, 2007 1st Annual. [S.l.], p. 3. 2007.

RAYMOND, E. **The Art of UNIX Programming**, Addison Wesley, 2003.

RESTful. **What Is REST?** Disponível em < http://www.restapitutorial.com/>. Acesso em 20 jan.2017.

RODRIGUES, J. M. **Evolução da Frota de Automóveis e Motos no Brasil**. Rio de Janeiro. 2013.

SOUZA, F. P. C. D. **Localização e leitura automática de caracteres alfanuméricos - Uma aplicação na identificação de veículos**. Escola de Engenharia, Universidade Federal do Rio Grande do Sul. [S.l.], p. 105. 2000.

SOUZA, F. P. C. D.; SUSIN, A. **SIAV - Um sistema de identificação automática de veículos**. XIII Congresso Brasileiro de Automática. Florianópolis, p. 4. 2000.

SUSIN, A. A.; TOZZI, C. L.; VON ZUBEN, F. J. **Melhorias para sistemas de reconhecimento da placa de licenciamento veicular**. Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas. [S.l.]. 2005.

SUTTER, H. **The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software**, 2009. Disponível em <http://www.gotw.ca/publications/concurrency-ddj.htm>. Acesso em 13 nov.2016.

TARTARI, A. L. **Implementação de medidas tecnológicas para suporte estratégico na gerência de garagens subterrâneas**. Escola de Engenharia, Universidade Federal do Rio Grande do Sul. [S.l.], p. 131. 2002.